# TEMPOS : une plate-forme pour le développement d'applications temporelles au dessus de SGBD à objets

Marlon Dumas Menjivar

# Université Joseph Fourier - Grenoble I

## Thèse

pour obtenir le grade de

Docteur de l'Université Joseph Fourier

(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Discipline : Informatique

présentée et soutenue publiquement par

Marlon Dumas Menjívar

le 26 Juin 2000

# Tempos : une plate - forme pour le développement d'applications temporelles au dessus de SGBD à objets

Composition du jury

| | |
|---|---|
| Président : | M. Michel Adiba |
| Rapporteurs : | Mme. Sophie Cluet |
| | M. Christian S. Jensen |
| Examinateur : | M. Jacques Le Maitre |
| Directeurs de thèse : | M. Pierre-Claude Scholl |
| | Mlle. Marie-Christine Fauvet |

Thèse préparée au sein du laboratoire LogicielsΓSystèmes et RéseauxΓIMAG

# Remerciements


Le temps qui change les êtres, ne modifie en rien
l'image que nous avons gardée d'eux

Marcel Proust
*Le temps retrouvé.*

Je voudrais cette page à la fois :

**Sincère** comme les remerciements que j'exprime à ceux qui ont contribué au déroulement et à l'achèvement de cette thèse.
À Marie-Christine Fauvet et Pierre-Claude Scholl, qui ont guidé et soutenu mes initiatives depuis de nombreuses années. Plus que des directeurs de thèse, ils ont souvent été des mentors et des amis.
À Michel Adiba qui a suivi avec attention mes premiers pas dans la recherche. Je le remercie au passage d'avoir accepté de présider le jury de cette thèse.
À Sophie Cluet qui a lu dans ce document non seulement les mots, les points et les virgules, mais aussi des idées si cachées et subtiles que j'ai du mal à comprendre comment elles ont pu se faufiler entre ces pages.
À Christian S. Jensen, qui m'a fait un très grand honneur en acceptant de rapporter cette thèse. Ses commentaires et critiques constructives m'ont apporté un regard éclairant sur les contributions et les erreurs de mes recherches.
*Thanks to Christian S. Jensen who honored me by reviewing this thesis. His comments and constructive criticisms provided me with a valuable viewpoint into my research's mistakes and contributions.*
À Jacques Le Maître pour l'intérêt qu'il a montré envers mes recherches et pour avoir accepté de participer au jury de cette thèse.
À Michel Scholl, aux conseils en or.
À tous ceux avec qui j'ai eu le plaisir de collaborer : Jean-François Canavaggio, Sonia Chardonnel, Chaouki Daassi, Laurence Nigay, Pierre Dumolard, Hervé Martin, Philippe Rigaux, ...
À tous les membres du groupe STORM et du laboratoire LSR.

**Profonde** comme l'amitié que je porte envers mes compagnons de route.
Rafael Lozano en premier lieu, mérite bien un paragraphe à part : collègue, colocataire, co-utilisateur d'$O_2$, co-auteur, et surtout, ami.
Un paragraphe à part aussi pour Sophie Dupuy.
Et un autre pour Genoveva et José-Luis.
Sans oublier bien sûr : Helena, Tony, Luciano, Edgar, les Mexicains de Grenoble, et ceux que j'oublie.

Et **éternelle** comme l'amour que je dédie à ma famille, qui tout au long de ces années a été aussi loin dans l'espace que près dans mon coeur.
*Mi reconocimiento eterno a mi familia que durante estos años ha estado tan lejos en el espacio como cerca en mi corazón.*
À Inga, ma patiente fiancée.

## Résumé

Les données temporelles sont présentes dans de nombreuses applications utilisant des Systèmes de Gestion de Bases de Données (SGBD). Aussi, la plupart de ces systèmes offrent des types correspondant aux concepts de date et de durée, grâce auxquels il est possible de modéliser des associations temporelles simples, comme par exemple la date de naissance ou l'âge d'une personne. Toutefois, à quelques nuances près, aucun de ces systèmes n'offre des abstractions dédiées à la modélisation d'associations temporelles plus complexes, telles que l'historique du salaire d'un employé, ou la séquence d'annotations attachées à une vidéo. Dès lors, ces associations doivent être codées au travers de constructeurs de type tels que "liste" et "n-uplet", et la sémantique de ce codage doit être intégrée dans la logique des programmes applicatifs, accroissant par là leur complexité. Pour combler ces lacunes, des extensions dites "temporelles" de modèles et de langages pour Bases de Données ont été proposées. Cette thèse analyse et unifie les contributions de ces travaux, dans le but de les intégrer dans une extension temporelle du standard pour SGBD à objets de l'ODMG. Le résultat est une plate-forme logicielle baptisée Tempos, fondée sur trois modèles de sophistication croissante: un modèle du temps, un modèle d'historiques et un modèle d'objets et de propriétés temporels. Ce dernier fournit des fonctionnalités facilitant la transformation de bases de données conformes à l'ODMG en des bases de données temporelles. À partir de ces trois modèles, des extensions des langages de spécification de schéma et d'interrogation de l'ODMG sont définies. Enfin, un outil de visualisation basé sur un nouveau paradigme de navigation interactive au travers d'objets temporels est développé. L'ensemble des propositions sont formalisées, implantées au dessus d'un SGBD commercial, et validées par des études de cas.



**Mots-clés:** bases de données temporelles, bases de données à objets, standard ODMG, langage de requêtes, visualisation de données.



## Abstract

Temporal data handling is a pervasive aspect of many applications built on top of Database Management Systems (DBMS). Accordingly, most of these systems provide datatypes corresponding to the concepts of date and span. These datatypes are adequate for modeling simple temporal associations such as the date of birth or the age of a person. However, they are insufficient when it comes to model more complex ones, such as the history of an employee's salary, or the sequence of annotations attached to a video. Since no datatypes dedicated to these kinds of associations are currently provided by DBMS, type constructors such as "list" and "tuple" should be used instead to encode them. The semantics of this encoding must then be integrated into the application programs, thereby increasing their complexity. To address this issue, several "temporal" extensions of data models and query languages have been proposed by the database research community. This thesis aims at contributing to these efforts by proposing such an extension in the context of the ODMG object database standard. The outcome is a platform for temporal applications development, namely Tempos, based on a family of three data models of increasing sophistication levels: a time model, a history model and a model for temporal objects and properties. This latter model is designed so as to ensure a seamless transition of applications accessing ODMG-compliant databases, towards temporal variants of them. Based on the above family of models, two extensions of ODMG's data definition and query languages are defined. Furthermore, a temporal object browsing technique based on a novel navigation paradigm is developed. The overall proposal has been fully formalized, implemented on top of an object DBMS, and validated through case-studies.



**Keywords:** temporal databases, object databases, ODMG standard, query language, data visualization.

# Table des matières

# Chapitre 1

# Introduction

Personne n'a vécu le passéΓpersonne ne vivra le futur; le présent est la forme de toute vie.

A. Schopenhauer
*Le monde comme volonté et représentation II.*

## 1.1 Contexte et objectifs

Les données temporelles sont présentes dans un grand nombre d'applications s'appuyant sur des Systèmes de Gestion de Bases de Données (SGBD)Γdepuis les plus classiques (systèmes d'information d'entreprises par exemple)Γjusqu'aux plus récentes : gestion de documents multimédiasΓsystèmes d'information environnementaleΓetc. En faitΓil est difficile de mettre en évidence des applications où le temps n'intervienne pasΓsous une forme ou sous une autreΓau niveau de la manipulation de données [Sno99].

AussiΓla plupart des SGBD fournissent des types de données pour dater des entités et des associations. Grâce à ces typesΓil est possible de modéliser des informations telles que la date de naissance d'une personne ou sa date d'embauche dans une entreprise. Selon les systèmesΓles fonctionnalités pour la manipulation de dates sont plus ou moins complètes à plusieurs égards : variété des modes d'expression et calendriers utilisablesΓvariété des niveaux de granularité et précision imposéeΓrichesse du jeu d'opérateursΓetc. À ce proposΓles standards SQL2 [Ins92] et ODMG (Object Database Management Group) [CB00] définissent un support minimal pour la représentation et la manipulation de dates et de durées exprimées par rapport au calendrier Grégorien.

ToutefoisΓà quelques nuances prèsΓaucun de ces systèmes n'offre des services permettant de maintenir et d'interroger l'historique des états d'une base de donnéesΓni des types modélisant l'historique des états d'une entité et/ou de ses associations. En conséquenceΓlorsque la prise en compte de l'évolution des données est requiseΓleur historique doit être codé à partir des types élémentairesΓet la sémantique de ce codage doit être intégrée dans la logique des

programmes applicatifs. Ces programmes se trouvent ainsi complexifiésΓet la connaissance que possède le SGBD sur la sémantique des données qu'il manipule réduiteΓce qui restreint la portée des optimisations qu'il est susceptible d'effectuer.

### 1.1.1 Bases de données temporelles

Le domaine des *Bases de Données Temporelles* vise à combler les lacunes mises en évidence ci-dessus : un modèle de données et un SGBD sont dits temporels s'ils offrent des concepts et des fonctionnalités permettant aux applications de dater les informations et de gérer leur historique. Notre projet de thèse s'inscrit dans ce contexteΓet vise plus spécifiquement à définir une plate-forme logicielle fournissant des abstractions de haut niveau pour la modélisation des aspects temporels de données encapsulées dans des objets. L'objectif de cette démarche est bien sûr d'offrir des services facilitant le développement d'applications manipulant ces objets.

De façon généraleΓle domaine des Bases de Données Temporelles traite de l'introduction du temps en tant que dimension à part entière dans les processus de modélisation et de manipulation de données. Depuis vingt ansΓce domaine de recherche a fait l'objet d'une activité très nourrieΓcomme en témoignent les nombreux ouvrages et conférences sur le sujet [RBL87ΓTCG+93ΓCT95ΓEJS98].

Une part importante de ces travaux est consacrée à la définition d'extensions "temporelles" de modèles de données et de langages de requêtes pour SGBD aussi bien relationnels qu'à objets. L'importance accordée à ces aspects provient de la complexité des contraintes d'intégrité et des expressions de requêtes sur des données temporelles [Sno99]. En effetΓsous certaines hypothèsesΓdes requêtes aussi simples à formuler en langue naturelle que *"Donner l'historique de la moyenne des salaires gagnés par l'ensemble des employés d'un magasin"*Γ font de l'ordre de trente lignes de code en SQL ou OQL.

Malgré l'importance de leurs contributionsΓnous pensons que la majorité des travaux autour des Bases de Données Temporelles souffrent d'un défaut récurrent : d'une façon ou d'une autreΓils visent à définir des modèles et des langages pour des SGBD dits "temporels"Γ qui se substitueraient aux SGBD actuels. Une telle hypothèse est non seulement illusoire vue la complexité de la technologie et du marché des SGBDΓmais en plusΓelle ignore les spécificités de chacune des applications ciblées.

En ce sensΓnous adoptons une approche différente. Notre objectif est de définir une plate-forme logicielleΓconstituée d'un modèle de données "stratifié" et d'un ensemble de services indépendantsΓpermettant la gestion de données temporelles à plusieurs niveaux de sophistication. Les applications peuvent ainsi choisirΓen fonction de leurs besoinsΓune strate du modèle de données et un ensemble de services. D'autre partΓles éventuels développeurs

de la plate-forme peuvent architecturer leur implantation soit comme une couche au dessus d'un SGBDΓsoit comme une extension au noyau de celui-ciΓselon le niveau de conformité visé et le degré d'extensibilité du système sous-jacent.

### 1.1.2 Bases de données à objets

Depuis plusieurs annéesΓles technologies à objets se sont imposées dans de nombreux domaines : environnements de programmationΓanalyse et conception de systèmes d'informationΓdéveloppement d'interfaces homme-machineΓconstruction d'applications répartiesΓetc. À cet égardΓles SGBD font exception. Depuis plus de vingt ansΓla plupart de ces systèmes reposent sur le célèbre modèle relationnel [Cod70]. Ce modèle se distingue par un système de types simpleΓun langage de requêtes déclaratif pour lequel de nombreuses techniques d'évaluation efficaces ont été élaborées [Gra93]Γet surtoutΓun standard bien établi [Ins92]. ToutefoisΓla simplicité de ce modèle le rend inadéquat dès qu'il s'agit de manipuler des données complexes. De plusΓson système de types est incompatible avec celui de la plupart des langages de programmation.

À côtéΓles SGBD à objetsΓcertes munis d'un système de types évoluéΓproche de celui des langages de programmation modernes tels que JavaΓne disposent que d'un standard (celui de l'ODMG) relativement instable et encore mal établiΓdont les versions successives [Cat94Γ CB97ΓCB00] comportent un certain nombre d'incohérences [Ala97ΓAla99]. De plusΓmalgré le nombre plus ou moins important de produits commerciaux se réclamant du label "SGBD à objets"Γpeu d'entre eux offrent toutes les caractéristiques attendues dans un SGBDΓet en particulierΓcelles concernant les performances et la robustesse [CC97].

Malgré ce constatΓforce est d'admettre que les SGBD à objets se sont imposés dans un certain nombre de domaines où le paradigme objet est central (CAOΓtélécommunicationsΓ bases de données scientifiques). De plus l'émergence de nouveaux besoins en matière de gestion de données complexesΓconjugués aux avancées en matière de techniques d'implantation de SGBD à objetsΓlaissent envisager une évolution favorable de ces systèmes dans le futurΓ justifiant ainsi des efforts de développement et de recherche tendant à améliorer les services qu'ils fournissent.

C'est dans cette optique que s'est inscrit le projet STORM (1995-99) [Adi96] au sein duquel la plate-forme décrite dans cette thèse a été développée. Ce projet visait à étudier les avancées en matière de bases de données à objets selon deux points de vue : (i) l'intégration de fonctions liées à la gestion de présentations multimédiasΓdes aspects dynamiques et temporels des objets ; (ii) la prise en compte de ces fonctions dans la conception d'applications.

Plus précisémentΓnotre projet de thèse fait suite à celui documenté dans [Can97]Γqui définit un modèle du temps et un modèle d'historiques fondés sur un système de types

fonctionnel à objets. C'est sur une version révisée et enrichie de ces modèles qu'est bâtie l'extension du standard de l'ODMG que nous proposons.

### 1.1.3 Bases d'objets temporels : l'approche TEMPOS

Le but principal de notre thèse est de définir des concepts et des fonctionnalités permettant de maîtriser la complexité du développement d'applications opérant sur des bases d'objets temporels, c'est-à-dire des objets dont la suite d'états au cours du temps est totalement ou partiellement observée.

Dans cette optique, nous abordons principalement trois aspects :

- Modélisation logique.
- Interrogation et mise à jour.
- représentation interne (stockage) et externe (manipulation visuelle).

Concernant le premier point, nous spécifions un modèle de données à objets conforme à l'ODMG, fournissant des types [1] pour la modélisation de valeurs temporelles (instants, durées, ensembles d'instants) et d'historiques. Ce modèle fourni aussi des extensions temporelles des concepts de classe et de propriété de l'ODMG. Ces extensions sont définies de manière à rendre transparente la migration d'applications développées au dessus de bases de données conformes à l'ODMG, vers des bases de données utilisant le modèle TEMPOS (Cf. § 1.2.2). Bien qu'inspirée de travaux réalisés précédemment dans le cadre d'extensions temporelles de modèles relationnels [SBJS98], ce dernier aspect constitue un point novateur de notre approche.

Les opérateurs définis sur les types du modèle ci-dessus, fournissent une interface de haut niveau pour l'interrogation de bases d'objets temporels. Toutefois, afin d'élever encore plus le niveau d'abstraction, et d'améliorer la lisibilité des formulations de requêtes temporelles, nous proposons un langage baptisé TEMPOQL. Conçu comme une extension d'OQL, ce langage traite les types temporels et historiques comme primitifs, au sens où leurs instances sont manipulées au travers d'opérateurs prédéfinis, chacun muni d'une syntaxe adaptée à ses caractéristiques. De plus, lorsque leur sémantique le permet, certaines constructions syntaxiques fournies par OQL sont surchargées pour s'appliquer à ces types. L'originalité majeure de TEMPOQL réside dans son respect de l'encapsulation des historiques. En effet, contrairement à la majorité des langages de requêtes pour données temporelles, TEMPOQL part du principe que l'interrogation d'historiques doit se faire autant que possible, sans passer par l'une de leurs représentations sous forme de collections (Cf. § 1.2.1).

La visualisation d'objets temporels, problématique peu étudiée dans le cadre des recherches sur les Bases de Données Temporelles, est abordée dans notre proposition au travers

1. Une grande partie de ces types et de leurs opérateurs ont été proposés dans [Can97].

d'une technique permettant de naviguer de façon orthogonale au travers de la dimension temporelle des objetsΓet au travers de leurs aspects structurels (Cf § 1.2.3).

Les contributions énumérées ci-dessus ont été regroupées dans une plate-forme appelée TEMPOSΓqui a été validée pendant et après son développement de diverses façons.

D'abordΓlorsque cela s'est révélé pertinentΓnous avons élaboré des spécifications formelles ou semi-formelles de certains de ses composants. C'est le cas par exemple des opérateurs sur les historiques pour lesquels nous fournissons des spécifications fonctionnellesΓet une traduction de ces spécifications en termes d'interfaces ODMG. Le langage de requêtes est quant-à-lui formalisé au travers de sa grammaireΓd'un ensemble de règles de typageΓet de schémas de traduction des expressions du langage vers les opérateurs du modèle du temps et d'historiques. EnfinΓla technique de visualisation est spécifiée sous forme de diagrammes conceptuels en UML [FS97]Γdécrivant aussi bien des aspects structurels (diagrammes de classes) que dynamiques (diagrammes de collaboration).

Le but de TEMPOS étant de fournir des fonctionnalités facilitant le développement d'applications temporellesΓnous avons aussi validé les propositions en les confrontant à divers domaines d'applications : applications de gestions classiquesΓétude d'activités et de déplacements d'individusΓet interrogation de documents vidéoΓentre autres. Ces expérimentations ont mis en évidence des abstractions et des fonctionnalités manquantesΓqui nous ont amenés à enrichir et réviser la plate-forme à tous ses niveaux.

EnfinΓdans le but de montrer la faisabilité de l'approcheΓnous avons effectué une validation du modèle au travers de l'implantation d'un prototype au dessus du SGBD $O_2$.

## 1.2 Aspects novateurs

Dans cette sectionΓnous introduisons trois des aspects novateurs de nos contributions :

- Le respect de l'encapsulation des représentations des historiques.
- La prise en compte de la migration d'applications vers des bases d'objets temporels.
- Le support pour l'exploration interactive de collections d'objets temporels.

En particulierΓnous montrons en quoi notre approche sur ces trois aspectsΓse distingue de celles adoptées dans des travaux similaires au notre.

### 1.2.1 Encapsulation des représentations des historiques

Une association temporelle est une donnée permettant de repérer un fait par rapport à une droite de temps. Les associations temporelles peuvent être classées en *instantanées* ou *duratives*[2]Γselon que les faits sont associés aux instants ou aux intervalles. Une association

2. Traductions des termes anglo-saxons *instant-based* et *interval-based* proposés dans [Cho94]

durative qualifie un fait par un intervalle (p. ex. *l'indice CAC 40 a augmenté de 80% entre 1995 et 1999*) sans impliquer pour autant que le fait a lieu à chaque instant dans cet intervalle (l'indice CAC 40 n'a pas augmenté de 80% chaque année entre 1995 et 1999 !). D'autre part, une association instantanée établit qu'un fait a lieu à un instant observé à une certaine granularité (p. ex. *tel magasin a vendu tant d'unités d'un produit donné le 15 Janvier 99*).

Par extension, les modèles de données temporelles peuvent aussi être classés en "instantanés" ou "duratifs" selon le type d'associations temporelles qu'ils reconnaissent [BBJS97]. TSQL2 [Sno95b] par exemple est un modèle de données temporelles "instantané" : lorsqu'on établit qu'un n-uplet appartient à une relation temporelle pendant un intervalle [i1..i2), ceci entraîne que ce n-uplet appartient à la relation à chaque instant entre i1 et i2 (i2 exclu). En revanche, SQL2 augmenté d'un type modélisant des intervalles peut être considéré comme duratif : rien dans sa sémantique ne permet de déduire que si un n-uplet est estampillé par un intervalle [i1..i2], l'information dénotée par ce n-uplet est "valide" à chaque instant entre i1 et i2. Certains modèles de données temporels proposés dans la littérature sont hybrides, soit parce qu'ils font persister des associations duratives mais qu'ils fournissent des opérateurs pour les transformer en des associations instantanées lors de l'interrogation (p. ex. IXRM [Lor93] et SQL/Temporal [SBJS98]) soit parce qu'ils font persister aussi bien des associations instantanées que duratives (p. ex. TOOBIS [TOO96b]). Il est intéressant de remarquer qu'aucun modèle de données temporelles proposé jusque là n'est exclusivement duratif. À notre avis, ceci provient du fait que la majorité des applications visées par le domaine de Bases de Données Temporelles manipulent des associations instantanées. En partant de cet argument, TEMPOS a été conçu comme un modèle temporel instantané.

Dans les modèles de données temporelles "instantanés", les associations temporelles élémentaires sont regroupées au sein de *relations temporelles* ou d'*historiques*. Une fois regroupées, ces associations peuvent être représentées de diverses façons. Par exemple, dans le cas d'une relation temporelle, chaque association élémentaire peut être représentée par un n-uplet estampillé par un instant [Tom98]. Une alternative consiste à regrouper plusieurs associations temporelles relatives à un même fait au sein d'un seul n-uplet, estampillé soit par un ensemble d'instants quelconque, soit par un intervalle. Cette dernière approche est de loin la plus répandue parmi les propositions de modèles temporels relationnels. Les opérateurs sur les relations temporelles sont alors définis sur cette représentation par intervalles, ce qui à notre avis rend leur formalisation inutilement lourde : en plus de définir le résultat attendu, la spécification de chaque opérateur doit aussi décrire la manière dont ce résultat est représenté. Pour la même raison, on observe dans ces modèles un écart important entre les expressions de requêtes en langue naturelle et celle dans le langage de requêtes qui leur est associé. Nous reviendrons plus en détail sur cette discussion au cours du document.

Pour ces raisons, nous nous positionnons contre l'approche consistant à figer une repré-

sentation des types temporelsΓque ce soit pour définir la sémantique des opérateurs sur ces typesΓou pour exprimer des requêtes faisant intervenir des instances de ces types. En effetΓcette approche non seulement donne lieu à des expressions de requêtes complexesΓmais elle est contraire au principe d'encapsulationΓà la base du paradigme à objets. Au lieu de celaΓnous pensons qu'un modèle d'objets temporels doit fournir des opérateurs définis indépendamment de toute représentationΓcorrespondant aux principaux types de raisonnement temporel : succession dans le tempsΓsimultanéitéΓchangement de granularitéΓetc.

Ces considérations s'appliquent également à la modélisation des autres types temporels (instantsΓintervallesΓensembles d'instantsΓetc.). En effetΓdans la plupart des modèles de données temporelles existantsΓles ensembles d'instants sont modélisés comme des ensembles d'intervalles disjoints et maximaux (connus sous le terme d'*éléments temporels* [WJSW98]). Même si cette représentation nous semble dans bien des cas adéquate au niveau physiqueΓ le fait de l'imposer dès le niveau logique est inutileΓet ne fait qu'introduire une séparation entre le concept modélisé et la construction du modèle de données correspondante.

En résuméΓnous pouvons dire qu'un modèle de données temporelles *respecte l'encapsulation des historiques* s'il fournit des opérateurs permettant d'exprimer les principaux types de raisonnement temporel et qu'il définit ces opérateurs indépendamment de toute représentation des associations temporelles. Au minimumΓle langage d'interrogation doit fournir un moyen d'exprimer des opérations de type sélectionΓprojectionΓjointureΓregroupement et agrégation sur les historiquesΓsans passer par les itérateurs correspondants sur les collections. De cette manièreΓun changement du choix de représentation interne des historiquesΓn'entraîne pas de changements dans les formulations des requêtes au sein desquelles ils interviennent.

### 1.2.2 Compatibilité ascendante et migration d'applications

La majorité des modèles et langages pour données temporelles proposés dans la littératureΓ sont en fait des extensions de modèles "non-temporels" (p. ex. le modèle relationnel ou celui de l'ODMG). Un avantage de cette approche réside dans le fait que les modèles résultants peuvent être intégrés au sein de systèmes existantsΓde telle sorte que les applications opérant au dessus de ces systèmes bénéficient des nouvelles fonctionnalités.

ToutefoisΓla migration transparente d'applications opérant sur un SGBDΓvers une extension temporelle de celui-ci impose certaines contraintes. En premier lieuΓl'extension ne doit pas altérer les services déjà fournis par l'ancien SGBD. Ce requisΓconnu sous le nom de *comptabilité ascendante*Γn'est pas spécifique aux extensions temporelles de SGBD mais s'applique à n'importe quel système censé remplacer un autreΓcomme par exempleΓles versions successives d'un microprocesseur ou d'un système d'exploitation.

Pour illustrer les conséquences du concept de compatibilité ascendante sur le modèle

TEMPOS, considérons l'exemple d'un SGBD conforme à l'ODMG, gérant une base de données sur les prêts de documents d'une médiathèque. La compatibilité ascendante établit que si ce SGBD est remplacé par une extension implantant le modèle TEMPOS, les programmes d'application accédant à la base de données en question restent opérationnels sans aucun changement.

Maintenant, supposons qu'une fois que tous les programmes hérités de l'ancien SGBD sont migrés vers le nouveau, il est décidé que les prêts effectués par chaque client doivent être historisés, mais de telle sorte que les programmes d'application hérités restent opérationnels (au pire modulo leur recompilation). En d'autres termes, vis-à-vis des programmes hérités, l'association entre les documents et les clients modélisant la notion d'emprunt doit rester fugitive[3], alors que vis-à-vis des nouveaux programmes l'association doit être temporelle.

Alors que la compatilibité ascendante peut être obtenue en bâtissant le modèle de données temporelles par strict ajout de concepts au modèle de départ, sans altérer ceux déjà existants, le support pour la migration d'applications vers des BD temporelles est plus difficile à satisfaire. D'ailleurs, [BBJS97] montre que pratiquement aucune extension temporelle de SQL (y compris TSQL2), ne satisfait cette propriété.

La même remarque s'applique aux extensions temporelles de modèle de données à objets, et en particulier à ceux fondés sur le standard ODMG (p. ex. T_ODMG [BFGM98] et TAU [KT96]). À titre d'exemple, sur l'application mentionnée ci-dessus, considérons une classe Document munie d'une propriété emprunté_par de type Adhérent. En TAU, si cette propriété est rendue temporelle par une évolution de schéma, alors tous les accès subséquents à cette propriété, retrouvent non seulement la valeur courante de la propriété (comme c'était le cas avant la modification de schéma), mais tout son historique. Les programmes d'applications qui considèrent cette propriété comme étant de type Adhérent ne sont donc plus opérationnels (Cf. figure 1.1).

Ce problème ne se pose pas en TOOBIS [TOO97] (une autre extension temporelle de l'ODMG). Toutefois, la migration d'applications non-temporelles en TOOBIS, se fait au détriment des applications temporelles. En effet, pour accéder à l'historique d'une propriété temporelle en TOQL (le langage de requêtes de TOOBIS), le nom de cette propriété doit être préfixée par l'un des mot-clés validⓉtransaction ou bitemporal. Ceci alourdit sensiblement l'écriture de requêtes temporelles. En fait, cette approche est d'une certaine manière, équivalente à celle consistant à dupliquer les symboles permettant de référencer une propriété, lorsque le type de celle-ci est modifié. Dans l'exemple de la bibliothèque cité ci-dessus, ceci veut dire qu'en TOOBIS tout se passe comme si après la modification de schéma, il existait deux propriétés dans le schéma, à savoir emprunté_par et valid_emprunté_par.

3. On dit qu'une association est *fugitive*, lorsqu'on ne s'interesse qu'à son état tel que défini par la dernière mise à jour, et *temporelle* lorsque ses états successifs sont datés et enregistrés.

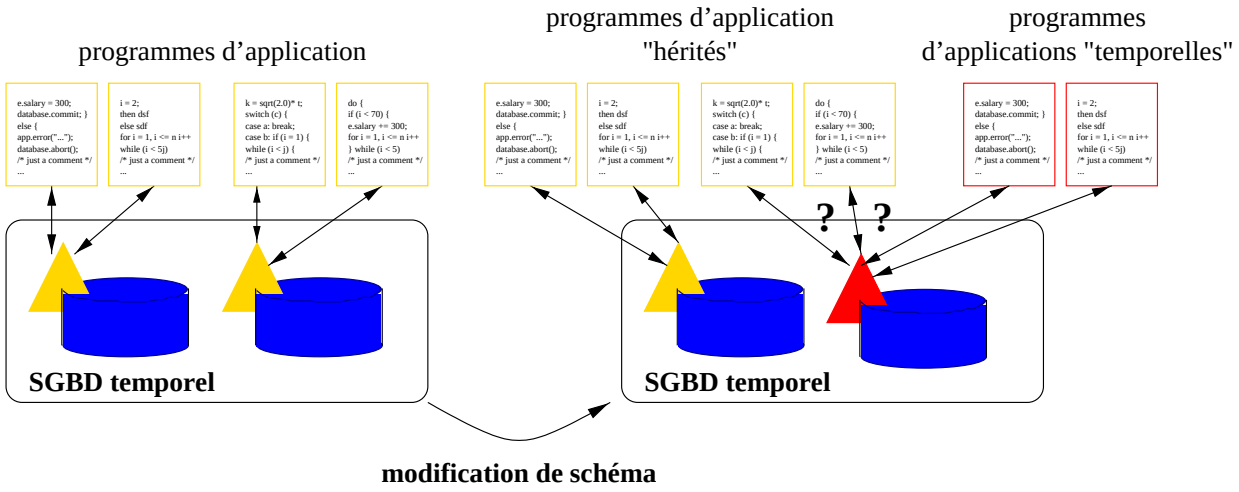



FIG. 1.1 – *Incompatibilité entre les applications et la base lors d'une évolution de schéma visant à historiser une partie des données. Dans ce dessin, les cylindres dénotent des bases de données et les triangles leur schéma. Les triangles jaunes (clairs) correspondent à des schémas non-temporels, alors que les triangles rouges (foncés) dénotent des schemas temporels. La base dont le schéma est modifié est celle qui se trouve à droite dans chaque côté du dessin.*

Nous proposons une approche différente : lorsqu'un schéma S est modifié dans le but de rendre "temporels" certains de ses composants (donnant alors lieu à un schéma S')Γles programmes d'application sont divisés en deux groupes : ceux opérant sur le schéma S et ceux opérant sur le schéma S'. Suivant une démarche classique dans le domaine de l'évolution de schéma [BF97]Γune vue de la base de données est introduite pour chacun de ces groupes d'applications (Cf. figure 1.2). Le modèle résultant est alors dit *bi-accessible*.

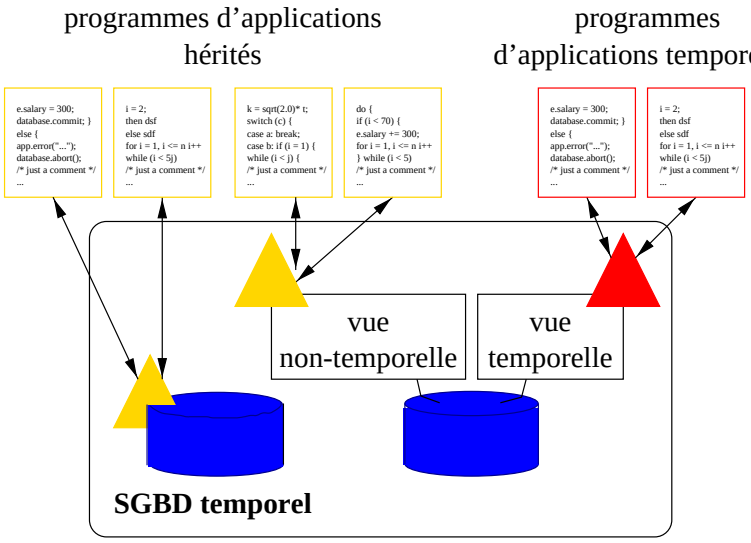



FIG. 1.2 – *Approche* TEMPOS *pour la migration d'applications vers de BD temporelles.*

### 1.2.3 Outils pour l'exploration de bases d'objets temporels

La multiplication de sources d'information, ajoutée au développement de modèles et d'outils de haut niveau pour leur interrogation, ont rendu au cours du temps l'usage des bases de données par des non-informaticiens, une pratique de plus en plus répandue. Aussi, demeurés longtemps l'apanage des spécialistes, les SGBD ont dû s'ouvrir à des nouveaux profils d'utilisateurs en intégrant dans leur offre, des langages d'interrogation visuels (p. ex. QBE [Zlo75]), ainsi que des outils pour la visualisation et la mise à jour interactive de données (p. ex. $O_2$Look [PBL+92]). Ces développements se sont nourris de nombreux travaux de recherche sur le sujet, dont on pourra trouver un état de l'art assez complet dans [CCLB97].

Alors que plusieurs langages de requêtes temporelles visuels ont été proposés [KG95, TPP95, FSC97], la visualisation de données a reçu peu d'attention dans le domaine des Bases de Données Temporelles. Ceci s'explique partiellement du fait que les recherches dans ce domaine se sont le plus souvent cantonnées à des modèles de données relationnels, où la problématique de la visualisation est souvent négligée à cause de la structure plate des données manipulées.

Dans le domaine de la Visualisation d'Information, de nombreuses techniques de représentation visuelle de séries temporelles ont été étudiées. La plupart de ces techniques sont dédiées aux séries temporelles quantitatives [Ber81, Tuf84], c'est-à-dire, des suites de nombres dénotant les valeurs prises par des variables numériques au cours du temps.

Certaines métaphores développées dans ces travaux ont été exploitées pour développer des techniques "généralistes" de visualisation d'objets temporels. Ainsi, [PMR+96] étudie une technique de visualisation de registres temporels fondée sur le concept de "ligne de temps", concept longuement étudié dans [Tuf84]. Dans [PMR+96], un registre temporel (c.à.d. un historique) est représenté par une suite de points ou de rectangles rangés horizontalement. Chacun de ces points (resp. rectangles) dénote un événement (resp. un état). La succession dans le temps est codée par l'arrangement de gauche à droite de la suite, et la métrique temporelle est transposée en une métrique sur l'espace visualisé. En d'autres termes : plus les points ou les rectangles sont séparés dans l'espace de la visualisation, plus les occurrences des événements ou des états qu'ils dénotent sont éloignées dans le temps. Enfin, des couleurs, des épaisseurs de traits différenciées, et des légendes textuelles sont utilisées pour différencier les événements et états entre eux, et pour donner à l'utilisateur une vision plus ou moins globale de la structure d'un registre, tout en lui permettant de se concentrer sur certains détails.

Notre approche concernant la visualisation de données dans le contexte d'une base d'objets temporels est la suivante. Nous distinguons dans l'acte de l'exploration visuelle d'une telle base, deux tâches :

- La navigation au travers des composants structurels des objets, c'est-à-dire : leurs attributs, leurs associations, et leurs liens de composition.

- L'observation de l'évolution des valeurs prises par ces composants au cours du temps.

Conformément, nous proposons une technique de visualisation interactive d'objets temporels qui traite ces deux tâches de manière orthogonale. L'idée générale de cette technique est de visualiser l'état à un instant dit *de référence*, d'un ensemble de chemins de navigation partant d'un objet temporel donné. Par ses interactions, l'utilisateur peut modifier soit l'ensemble de chemins de navigation visualisés, soit l'instant de référence; ces opérations correspondant exactement aux deux tâches énumérées ci-dessus.

## 1.3 Organisation du document

Le corps de cette thèse commence par un aperçu de la technologie offerte par les SGBD à objets, et de ses limitations au regard de la manipulation de données temporelles (chapitre 2). Ce chapitre contient par ailleurs une partie importante de notre description de l'état de l'art en matière de Bases de Données Temporelles, même si celle-ci est en fait répartie sur l'ensemble du document. En effet, nous avons choisi de présenter les approches existantes au fur et à mesure que nous les comparons à la nôtre.

Dans le chapitre 3, nous décrivons le modèle d'objets temporels TEMPOS, sur lequel se basent les outils pour l'interrogation et la visualisation que nous présentons dans le chapitre 4. La section 3.1 décrit les élements de [Can97] nécessaires à la compréhension de cette thèse. Le reste de ces deux chapitres comporte l'essentiel de nos contributions.

Le chapitre 5 est consacré à la description de deux applications auxquelles TEMPOS a été confronté. L'accent est mis sur d'une part sur les apports de ces applications à la validation et à l'enrichissement du modèle, et d'autre part sur les aspects de chaque application qui n'ont pas été abordés du fait des limitations de notre approche.

L'architecture du prototype est décrite dans le chapitre 6. Cette description est accompagnée d'une discussion autour des représentations physiques des instances du type historique et de l'implantation de certains des opérateurs définis sur ce type. Par ailleurs, nous détaillons quelques problèmes rencontrés lors de l'implantation du modèle TEMPOS et de TEMPOQL.

Le chapitre 7 conclut par un bilan du travail et une analyse de ses perspectives.

Enfin, les annexes comportent des spécifications formelles de quelques éléments de notre proposition.

# Chapitre 2

# Position du problème

Dans ce chapitre, nous discutons de quelques aspects de la technologie fournie par les SGBD à objets et de ses limitations au regard de la manipulation de données temporelles.

Nous faisons ensuite une synthèse de quelques solutions à ces limitations étudiées dans des travaux de recherche autour des Bases de Données Temporelles. Cette synthèse couvre aussi bien des travaux se plaçant dans le cadre d'une approche à objets, que d'autres destinés à étendre des modèles relationnels. En effet, certains concepts développés dans ces derniers travaux, s'appliquent avec quelques changements aux modèles à objets.

Tout au long de ce chapitre, comme d'ailleurs dans le suivants, nous supposons que le lecteur possède quelques notions de base autour des SGBD relationnels et des langages de programmation à objets.

## 2.1 SGBD à objets

### 2.1.1 Cadre général

Du point de vue fonctionnel, un SGBD est un ensemble d'outils plus ou moins intégrés, fournissant des services liés au stockage de données structurées ou semi-structurées en mémoire secondaire, éventuellement distribuée. Parmi ces services, nous nous intéressons à ceux concernant la spécification des contraintes structurelles et sémantiques des données (c'est-à-dire leur *schéma*), et à ceux qui permettent leur mise à jour et leur interrogation.

Ces deux types de services sont accessibles au travers de langages fondés sur un *modèle de données*, qui fournit en particulier un *système de types* et un *modèle de persistance*. La définition du schéma s'effectue le plus souvent dans un langage déclaratif appelé le Langage de Définition de Données (LDD). Ce langage permet en particulier, de spécifier les types, et de déclarer des variables persistantes dont les valeurs appartiennent à ces types. L'interrogation et la mise à jour quant à elles, s'effectuent au travers d'un ou plusieurs langages déclaratifs

ou impératifsΓparfois mélangésΓdont la compilation (ou l'évaluation) est dans certains cas confiée à un module extérieur. Plus précisémentΓdans les SGBD relationnels les accès aux données se font au travers de langages déclaratifs (des variantes de SQL par exemple)Γdont le processus d'évaluation est complètement intégré dans le SGBD. A contrarioΓdans les SGBD à objets les accès aux données se font principalement au travers de langages de programmation impératifs à objets (dont la compilation est effectuée par un logiciel extérieur) couplés à un langage d'interrogation déclaratif (dont l'évaluation est à la charge du SGBD).

Cette diversité des interfaces d'accès aux SGBD à objets est source de nombreux problèmes lors de la définition du modèle de données sous-jacent. En effetΓcelui-ci doit offrir un système de types transversal à plusieurs langages de programmation généralement incompatibles. Par ailleursΓdu fait que le SGBD n'est pas entièrement maître du processus de compilation de ces langagesΓla gestion de la persistance et l'optimisation des accès à la mémoire secondaireΓposent de nombreux problèmes.

Ces difficultés techniques ont retardé l'émergence d'un consensus effectif autour des concepts et fonctionnalités attendus dans un SGBD à objets. Au début des années 90Γun standard dans ce domaine a été défini par l'ODMG (Object Database Management Group) [Cat94]Γorganisme regroupant plusieurs éditeurs et utilisateurs de SGBD à objets. ToutefoisΓce standard n'a été intégré que très partiellement dans les produits existants. Par ailleursΓsa stabilité a été fortement affectée par ses multiples révisionsΓrévisions motivées d'une part par les incohérences que comportaient ses premières versions [Ala97]Γet d'autre partΓpar les changements technologiques des années 90Γet en particulier l'avènement du langage Java.

Ce manque de consensus effectifΓ conjugué à la concurrence des technologies voisines (SGBD relationnels étendus par exemple) ont longtemps freiné le développement des SGBD à objets d'envergure industrielle. AinsiΓannoncés en fanfare à la fin de années 80Γles SGBD à objets ont parfois été signalés comme des fiascos industriels vers le milieu des années 90 [CC97]. Toujours est-il qu'un bon nombre de ces systèmes se sont imposés dans des domaines où la technologie objet est centrale (CAOΓbases de données techniquesΓgestion des processusΓetc.)Γont percé sur des marchés "niches" tels que les bases de données scientifiques et les télécommunicationsΓet dans une moindre mesureΓsont aussi utilisés sur des applications plus classiques comme les bases de données financières.

La technologie et le marché ayant atteint un certain degré de maturité et de stabilitéΓles éditeurs de SGBD à objets commencent à converger vers des approches similaires. À présentΓ la plupart des produits commerciaux concernés offrent des passerelles vers les langages C++ et Java plus ou moins conformes à l'ODMG. Par ailleursΓcertains de ces produits (Poet[1] et Jasmine[2] par exemple) incluent des implantations assez complètes du langage d'interrogation

1. http://www.poet.com
2. http://www.cai.com/products/jasmine.htm

OQL de l'ODMG. Enfin, des outils de type "médiateur" fournissant des modèles et des services conformes à l'ODMG au dessus de SGBD relationnels ont récemment vu le jour (Cf. ObjectDRIVER[3] et JavaBlend[4]).

Aussi, même si le standard proposé par l'ODMG tarde à s'imposer, son modèle de données, son langage de requêtes, et les passerelles vers des langages de programmation qu'il définit restent représentatifs des approches utilisées, et constituent donc un point de repère pour la spécification et le développement de nouvelles fonctionnalités au sein des SGBD à objets. Pour cette raison, nous avons décidé de prendre certains composants de ce standard comme modèles de référence pour notre travail. L'objet des prochains alinéas est de présenter brièvement ces composants, afin de fixer une partie des concepts et notations utilisés par la suite.

### 2.1.2 Le standard ODMG

Le standard ODMG dans sa version 2.0 [CB97] est structuré en plusieurs composants s'appuyant sur un unique *modèle d'objets*. Ce modèle définit en particulier un système de types à base de classes avec les caractéristiques suivantes :

- Une dichotomie est faite entre les *littéraux* (objets immuables) et les *objets* proprement dits. Ces derniers possèdent un identificateur et un état qu'il est possible de modifier.
- Chaque objet a un type. Les éléments d'un type ont le même ensemble d'états possibles et le même *comportement* (ensemble d'*opérations*).
- L'état d'un objet est structuré en des *propriétés*. Le concept de propriété regroupe celui d'*attribut* tel qu'il apparaît dans la plupart des langages de programmation à objets et celui d'*association* au sens des modèles entite-association. Trois types d'associations (1-1, 1-N et N-M) sont supportées. Une association apparaît concrètement sous forme de deux propriétés *inverses* (appelés des *chemins de traversée*) attachées aux classes participantes. Des contraintes d'intégrité référentielles s'appliquent aux chemins de traversée définissant une association.

  Pour illustrer ces propos, on considère le schéma suivant :

```
class Employé {
   attribute short salaire; /* short est un type ODMG modélisant des entiers. */
   relationship Département département inverse Département::employés;
   /* département est une propriété de type Département. Plus précisément, il s'agit
   d'un chemin de traversée d'une association dont le chemin inverse est la propriété
   de nom employés de la classe Département */
}
```

3. http://www.infobjects.net
4. http://www.sun.com/software/javablend

```
class Département {
   relationship set<Employé> employés inverse Employé::département;
}
```

La propriété salaire est un exemple d'attributΓalors que les propriétés employés et département sont des chemins de traversée. Ces deux chemins de traversée définissent une relation 1-N entre les classes Employé et Département. Les objets de ces deux classes sont liés par les contraintes suivantes :

(1) e de type Employé $\Rightarrow$ e $\in$ e.département.employés
(2) d de type Département $\wedge$ e $\in$ d.employés $\Rightarrow$ e.departement = d

- Le comportement des objets d'une classe est défini par un ensemble d'opérations. Les opérations possèdent une signature et une implantation.
- Propriétés et opérations sont classifiées en *privées* et *publics* avec leurs sens habituels.
- Une distinction est faite entre *interface* et *classe*. Une interface correspond à la spécification abstraite d'un type sous forme d'un ensemble de propriétés et d'opérations (sans le code associé). Une classe porteΓoutre cette composante abstraiteΓune implantation pour chacune des opérations qu'elle définit etΓdans certains casΓune *extension* comportant l'ensemble de ses instances persistantes. Les interfaces ne comportent jamais d'extensionΓpuisqu'elles ne sont pas instanciables. Les interfaces sont organisées en une hiérarchie d'héritage (suivant une sémantique de sous-typage)Γalors que les classes le sont en une hiérarchie d'*inclusion*. Le passage des interfaces aux classes est modélisé par une relation d'*implantation* (une classe *implante* une ou plusieurs interfaces).

Parmi les principaux composants du standard on trouve :

- ODL (Object Definition Language)Γ un langage déclaratif pour la spécification du schéma initial d'une base de données. La consultation et la modification ultérieure des schémas ainsi définis se font au travers d'un autre composant : le *gestionnaire de méta-données*.
- OIF (Object Interchange Format)Γun format de représentation d'objets sous forme textuelleΓdestiné à faciliter l'échange de données entre SGBD.
- OQL (Object Query Language)Γun langage de requêtes associatif "à la SQL" dont nous donnons un aperçu dans le § 2.1.3.

Par ailleursΠle standard définit des *passerelles* d'accès pour les langages de programmation SmallTalkΓC++ et Java. Le but de ces passerelles est de rendre accessibles aux applications codées dans ces langagesΓtous les services offerts par le SGBD (persistanceΓaccès transactionnelΓconcurrent et distribué aux objetsΓetc.)Γtout en limitant les extensions apportées à

la syntaxe du langage. Concrètement, chacune de ces passerelles comprend au moins :

- Une correspondance entre les types prédéfinis du modèle d'objets de l'ODMG, et les types fournis par le langage de programmation.
- Un ensemble de classes destinées à modéliser les concepts de *session*, *transaction*, *requête*, etc., ainsi que quatre types de collections : ensembles, multi-ensembles, listes et tableaux.
- Une extension du sous-langage de définition de types du langage de programmation concerné, permettant d'indiquer les classes qui doivent persister, les contraintes d'intégrité référentielles et de clé, le nom des extensions des classes, etc.

Le modèle d'objets de l'ODMG est donc le support de deux langages déclaratifs, et de trois langages impératifs fondés sur des systèmes de types incompatibles. Certaines de ces incompatibilités se reflètent en des incohérences entre les composants du standard comme le montre [Ala97]. Une bonne partie de ces incohérences proviennent du fait que le langage C++ supporte le concept de *classe paramétrée*, alors que SmallTalk et Java (dans sa spécification actuelle) ne le supportent pas du tout, et qu'OQL et ODL le supportent mais de façon limitée. Or, les classes paramétrées sont fondamentales pour assurer le typage statique des opérations sur les collections. D'autres incohérences concernent le modèle de persistance, qui mélange le concept de persistance transitive (si un objet persiste, alors tous les objets qui lui sont accessibles persistent aussi), avec celui de persistance par classes (pour qu'un objet persiste, il doit être instance d'une classe persistante). Cette dualité mène à des situations sinon incohérentes, du moins génératrices d'erreurs à l'exécution.

En partant de ce constat, [Ala99] introduit une famille de trois modèles de données "à la ODMG", comme alternative au modèle unique proposé dans l'état actuel du standard. Les trois membres de cette famille sont d'un niveau de complexité croissant, de manière à permettre la définition d'un standard avec plusieurs *niveaux de compatibilité*, concept qui manque cruellement dans la version actuelle de l'ODMG. Nous présentons ci-dessous les deux premiers membres de cette famille.

Le premier d'entre eux reprend le système de types de Java, et correspond plus ou moins au modèle de données sous-jacent à la passerelle Java de l'ODMG. Il s'agit donc d'un modèle sans classes paramétrées ni héritage multiple, qui fait une distinction entre interfaces et classes. Le modèle de persistance est transitif et orthogonal au système de types : n'importe quel objet peut être attaché à une variable persistante.

Le deuxième modèle est très similaire au premier, à l'exception qu'il rajoute le concept de *classe paramétrée contrainte*. Par rapport aux classes paramétrées simples (celles qu'on trouve en C++), les classes paramétrées contraintes permettent de restreindre l'ensemble de types acceptés en paramètre. À titre d'exemple, supposons qu'on ait à définir un composant logiciel

réutilisable pour la manipulation de listes triées. Dans le formalisme des classes paramétrées contraintes, un tel composant se modélise naturellement par une classe paramétrée dont la partie intentionnelle est spécifiée comme suit :

```
interface Comparable {
  public boolean Inférieur(Comparable autre);
}
interface CollectionOrdonnée<T : Comparable> : Collection { ... }
```

Dans cet exemple, le paramètre du générateur d'interfaces CollectionOrdonnée est contraint à être un sous-type de Comparable, et possède donc une méthode Inférieur supposée implanter une relation d'ordre linéaire et total.

### 2.1.3 Le langage de requêtes OQL

Conçu à l'origine comme l'un des composants du SGBD $O_2$ [BDK92], OQL est un langage de requêtes à objets fondé sur un paradigme fonctionnel [Clu98] : les requêtes s'expriment par composition d'opérateurs, en partant des noms persistants définis dans le schéma de la base et/ou de constantes. Parmi les opérateurs du langage figurent :

- Des constructeurs de collections et de structures. Par exemple list(1, 3) est une valeur de type "liste d'entiers", alors que struct(a : 2, b : 2) est une valeur de type structure, possédant deux attributs de type entier.
- Un opérateur de création d'une nouvelle instance d'une classe. À ce propos, le standard ne spécifie pas si les objets ainsi créés sont ajoutés à l'extension de leur classe et deviennent donc persistants, dans quel cas il s'agirait d'un opérateur de modification.
- Des opérateurs arithmétiques et de comparaison.
- Les opérateurs booléens not, and, or, et les variantes séquentielles de ces derniers : andthen et orelse.
- L'opérateur "." (synonyme de "->"), permettant d'accéder à la valeur d'une propriété, et d'effectuer des appels de méthodes. Le standard ne spécifie pas si les méthodes ainsi invoquées peuvent effectuer des modifications.
- Des opérateurs permettant d'accéder à un élément d'une liste ou d'un tableau par son rang, ou d'extraire une partie d'une liste.
- Des opérateurs d'agrégation de collections de valeurs numériques (les mêmes qu'en SQL2).
- Les quantificateurs existentiels et universels (exists et forall).
- Des itérateurs sur des collections : produit cartésien, sélection, application d'une fonction à chaque élément d'une collection, partitionnement et tri. Ces itérateurs sont regroupés dans une famille de constructions syntaxiques de la forme select ... from ...

where ... group by ... having ... order byΓavec un certain souci de ressemblance avec SQL. À titre d'exempleΓsoit LesEmployés une collection d'objets de la classe Employé définie ci-dessus. La requête suivante retrouve l'ensemble de départements où il y a au moins un employé avec un salaire supérieur à 5000.

```
select distinct e.departement
from LesEmployés as e /* la variable e itère sur la collection LesEmployés */
where e.salaire > 5000
```

Le langage OQL combine donc des opérateurs sur des littéraux et des objetsΓavec des opérateurs d'ordre supérieur tels que les itérateursΓpour lesquels certains des paramètres sont des fonctions elles-mêmes spécifiées en OQL. Notons que l'appel de méthodes peut aussi être vu comme un opérateur d'ordre supérieurΓdans lequel la fonction paramètre est décrite dans un langage de programmation externe.

Plusieurs sémantiques formelles d'OQL ont été proposées. [Clu91] par exempleΓfournit une sémantique opérationnelle des itérateurs sur les collectionsΓpar traduction en des schémas de programmes. [FM95] propose une formalisation en termes de compréhensions de monadesΓ formalisme de type fonctionnel assez répandu dans la théorie des catégories. Cette dernière formalisation a été utilisée dans le développement d'au moins deux optimiseurs d'OQL : celui implanté dans le système lambda-DB[5]Γet celui développé par le projet CROQUE[6]. EnfinΓ [RS97] fournit une formalisation détaillée des règles de typage d'OQLΓainsi qu'une traduction vers un calcul dont la sémantique par contre n'est pas explicitée.

L'optimisation d'OQL a été longuement étudiée (voir p. ex. [Clu91ΓCM93ΓFeg98]). Pour l'essentielΓces travaux combinent des techniques d'optimisation éprouvées dans le cadre de SQLΓavec des techniques de normalisation dans lesquelles on cherche à éliminer les emboîtements d'itérateurs. D'autres optimisations permettent d'exploiter la présence éventuelle d'index de type arbre BΓpar réécriture des opérations de navigation en des jointures. Par ailleursΓ[Clu91] considère des techniques d'optimisation fonctionnelleΓcomme par exemple celles fondées sur la mémorisation de résultats d'appel de fonctionsΓdans le but d'éviter son évaluation répétée. À notre connaissanceΓaucune de ces techniques n'aborde de façon approfondie le problème de l'optimisation "globale" des requêtesΓoù la partie de la requête écrite en OQL et celle "cachée" dans les appels de méthodesΓseraient optimisées ensemble.

En résuméΓon retient trois points : (i) OQL est un langage de requêtes fonctionnel à objets pour lequel des sémantiques formelles existentΓ(ii) les requêtes écrites dans ce langage se prêtent à de nombreux types d'optimisationΓmais malheureusement (iii) ces optimisations ne concernent pas le code des méthodes qui y sont éventuellement invoquées.

5. http://lambda.uta.edu/lambda-DB
6. http://www.fmi.uni-konstanz.de/~gluche/croque

## 2.2 Développement d'applications temporelles au dessus de SGBD à objets

Dans cette section, nous illustrons sur une étude de cas, certaines limites de l'approche "SGBD à objets" vis-à-vis de la gestion de données comportant des associations temporelles. Cette discussion reprend des idées que nous avons développées dans [DFS00].

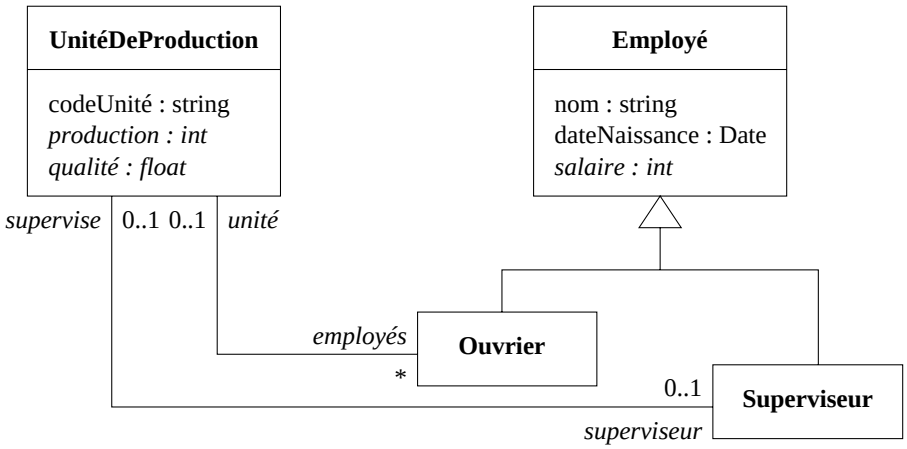
L'étude de cas retenue concerne le suivi dans le temps des unités de production d'une usine et des employés qui leur sont affectés. Une unité possède un code qui l'identifie de façon unique. On connaît par ailleurs les historiques du volume et de la qualité de ses productions journalières. La qualité de la production d'une unité est représentée par un réel compris entre 0 et 1, appelé par la suite le *facteur qualité*. Pour chaque employé, on retient son nom, sa date de naissance, et l'historique de son salaire horaire observé à la granularité du jour. Les employés sont classifiés en ouvriers et superviseurs (classification non-exhaustive). À chaque instant, un superviseur est affecté à au plus une unité, et chaque unité est supervisée par au plus un superviseur. Un ouvrier de son côté, travaille dans au plus une unité de production, et une unité de production est associée à zéro, un ou plusieurs ouvriers.

### 2.2.1 Modélisation des données

La figure 2.1 fournit une modélisation d'une vision instantanée des données de l'application, d'abord sous forme d'un diagramme de classes UML [FS97], ensuite sous forme de déclarations en ODL. Les attributs et rôles pour lesquels l'historique est observé sont indiqués par l'utilisation d'une police *italique*.

Le passage de cette modélisation "instantanée" à une modélisation prenant en compte l'historique des données dépend des modalités d'évolution de chacun des attributs et rôles concernés. Dans le cas de l'attribut production, pour lequel il est naturel de penser que la valeur varie presque tous les jours, on peut envisager une représentation sous forme de collection de couples ⟨ date, valeur ⟩. Dans le cas de l'attribut salaire, cette représentation est aussi envisageable, mais étant donné que le salaire horaire d'un employé ne varie pas souvent, on lui préfère des représentations à base de couples ⟨ intervalle, valeur ⟩. D'autres représentations sont aussi envisageables, et le choix parmi elles est une décision souvent délicate.

Le diagramme de classes de la figure 2.2(a), sur lequel nous nous appuyons par la suite, n'est donc qu'une alternative parmi d'autres. On remarque que ce diagramme est sensiblement plus complexe que celui correspondant à la vision "instantané" des données. Cette complexité provient du fait que le diagramme doit décrire, en plus de la sémantique des données, leur représentation. Nous reviendrons sur ce point dans le chapitre 5, où nous proposerons un moyen de maîtriser cette complexité à l'aide d'un stéréotype UML.




(a) Diagramme de classes UML.

```
class UnitéDeProduction(extent LesUnités, key codeUnité) {
   attribute string codeUnité;
   relationship Superviseur superviseur inverse Superviseur::supervise;
   relationship set<Ouvrier> ouvriers inverse Ouvrier::unité;
   attribute int production;
   attribute float qualité;
}
class Employé (extent LesEmployés, key (nom, dateNaissance)) {
   attribute string nom;
   attribute Date dateNaissance;
   attribute float salaire;
}
class Ouvrier extends Employé (extent LesOuvriers) {
   relationship UnitéDeProduction unité inverse UnitéDeProduction::ouvriers;
}
class Superviseur extends Employé (extent LesSuperviseurs) {
   relationship UnitéDeProduction supervise;
}
```

(b) Traduction du diagramme de classes en ODL.

FIG. 2.1 – *Modélisation d'une vision "instantanée" des données de l'application. Les noms d'attributs et d'associations en italiques correspondent aux données qu'on veut "historiser".*

Le diagramme UML de la figure 2.2(a) conduit naturellement au schéma ODL spécifié dans la figure 2.2(b). Dans ce schémaΓles valeurs des attributs historiques sont des collections de couples ⟨ estampille temporelleΓvaleur ⟩. Dans la suiteΓchacun de ces couples est appelé un *instantané*Γet ses composants sont respectivement appelés *valeur temporelle* (vt) et *valeur structurelle* (vs).

### 2.2.2 Contraintes d'intégrité

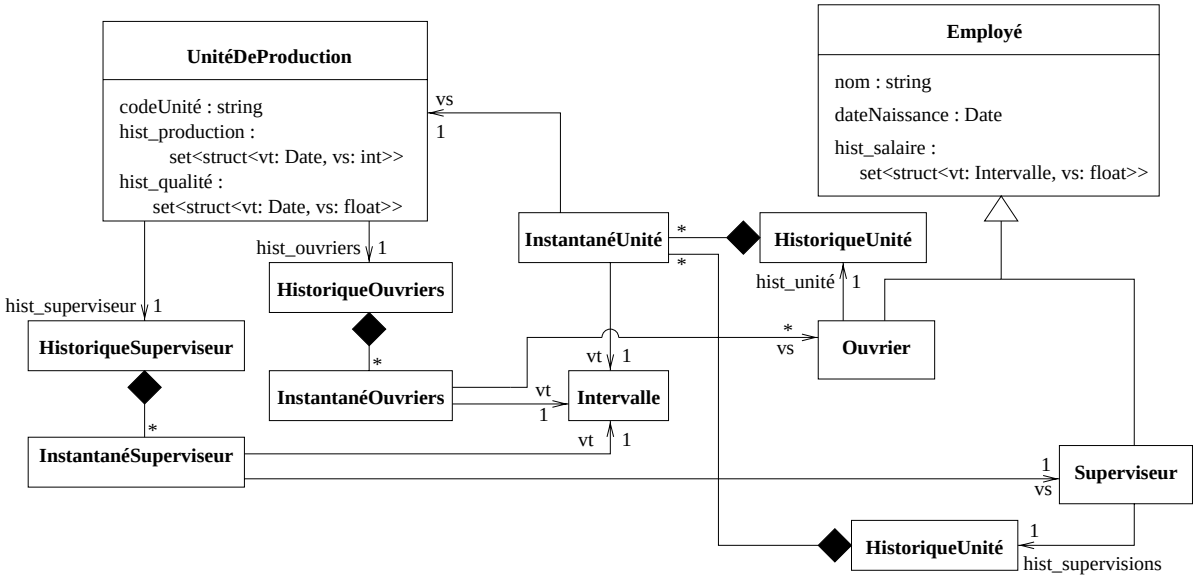
Tels quelsΓle diagramme UML et le schéma ODL de la figure 2.2 admettent plusieurs types d'incohérences. Par exempleΓil est possible d'avoir dans l'historique des salaires d'un employé deux instantanés de la forme ⟨ [1/1/99, 25/3/99], 10000 ⟩ et ⟨ [1/3/99, 31/8/99], 12000 ⟩Γce qui voudrait dire qu'entre le 1/3/99 et le 25/3/99Γl'employé en question gagnait à la fois 10000 et 12000 ! Ce type d'incohérences peut être évité en introduisant des contraintes sur les valeurs temporelles des instantanés. Plus précisémentΓétant donnés deux instantanés appartenant à un même historiqueΓon impose que leurs valeurs temporelles ne s'intersectent pas (s'il s'agit d'intervalles) ou qu'elles soient différentes (s'il s'agit d'instants).

Par ailleursΓle diagramme UML et le schéma ODL de la figure 2.2 permettent de représenter une même information temporelle de plusieurs manières. AinsiΓl'information *"entre le 1/3/99 et le 31/8/99, l'employé de nom X a pour salaire 12000"*Γpeut être représentée soit par un seul instantané dans l'historique des salaires de cet employéΓà savoir ⟨ [1/3/99, 31/8/99], 12000 ⟩Γsoit par deux instantanés : ⟨ [1/3/99, 31/5/99], 12000 ⟩ et ⟨ [1/6/99, 31/8/99], 12000 ⟩. Dans le pire des casΓcette information peut être disséminée sur autant d'instantanés qu'il y a de jours entre le 1/3/99 et le 31/8/99. Afin d'éviter cette situationΓil convient d'introduire une contrainte supplémentaire sur les attributs historiques représentés par intervallesΓà savoir : si la valeur historique d'un tel attribut comporte deux instantanés avec la même valeur structurelleΓalors leurs valeurs temporelles ne se jouxtent pas[7]. Cette contrainte est appelée par la suite *contrainte de maximalité des attributs historiques représentés par intervalles.*

Remarquons queΓcontrairement à la modélisation de la vision "instantané" de la base de données exemple (figure 2.1)Γla modélisation de la vision "historique" (figure 2.2) ne tient pas compte de l'intégrité référentielle entre les classes SuperviseurΓUnitéDeProduction et Ouvrier. Autrement ditΓles contraintes suivantes ne sont modélisées ni dans le diagramme UML de la figure 2.2(a)Γni dans le schéma ODL correspondant :

- À tout instantΓle superviseur d'une unité supervise cette unité.
- À tout instantΓsi un ouvrier est affecté à une unité de productionΓalors il appartient à l'ensemble d'ouvriers de cette unité.

7. Deux intervalles se *jouxtent* si leur intersection est vide et si leur union est un intervalle. Par exemple l'intervalle [1..4] jouxte [5..8].




(a) Diagramme de classes UML.

```
typedef struct { Date binf; Date bsup; } IntervalleDates;
class UnitéDeProduction(extent LesUnités, key codeUnité) {
   typedef struct { Superviseur vs; IntervalleDates vt; } InstantanéSuperviseur;
   typedef struct { set<Ouvrier> vs; IntervalleDates vt; } InstantanéOuvriers;
   typedef struct { short vs; Date vt; } InstantanéProduction;
   typedef struct { float vs; Date vt; } InstantanéQualité;
   attribute string codeUnité;
   attribute set<InstantanéSuperviseur> hist_superviseur;
   attribute set<InstantanéOuvrier> hist_ouvriers;
   attribute set<InstantanéProduction> hist_production;
   attribute set<InstantanéQualité> hist_qualité;
}
class Employé (extent LesEmployés, key (nom, dateNaissance)) {
   typedef struct { float vs; IntervalleDates vt; } InstantanéSalaire;
   attribute string nom;
   attribute Date dateNaissance;
   attribute set<InstantanéSalaire> hist_salaire;
}
class Ouvrier extends Employé (extent LesOuvriers) {
   typedef struct { UnitéDeProduction vs; IntervalleDates vt; } InstantanéUnité;
   attribute set<InstantanéUnité> hist_unité;
}
class Superviseur extends Employé (extent LesSuperviseurs) {
   typedef struct { set<Unité> vs; IntervalleDates vt; } InstantanéUnité;
   attribute set<InstantanéUnité> hist_supervisions;
}
```

(b) Traduction du diagramme de classes en ODL.

FIG. 2.2 – *Modélisation de la vision "historique" des données de l'application*

Dans le cas du diagramme UML, toutes les contraintes ci-dessus peuvent être intégrées sous forme d'annotations, exprimées par exemple en OCL (Object Constraint Language). En ODL par contre, il n'existe pas de mécanisme permettant de définir des contraintes d'intégrité autres que les contraintes de clé et les contraintes référentielles classiques. En conséquence, aucune de ces contraintes ne peut être assurée par le SGBD et elles doivent donc être prises en compte au niveau des programmes d'application. Cette situation est assez regrettable, d'autant plus que le renforcement de ces contraintes se révèle difficile.

### 2.2.3 Expression de requêtes

L'une des principales motivations de cette thèse trouve sa source dans le constat suivant : les requêtes faisant intervenir des associations temporelles sont particulièrement difficiles à exprimer dans les langages de requêtes existants (p. ex. OQL).

Afin d'illustrer la nature de cette complexité, nous considérons dans un premier temps des requêtes où il s'agit de mettre en correspondance les valeurs prises par plusieurs historiques "aux mêmes instants" (on parlera de *valeurs synchrones*). Un tel besoin se manifeste dans au moins deux cas de figure :

- Lorsqu'il s'agit de comparer deux ou plusieurs historiques, ou d'obtenir un nouvel historique par combinaison d'autres.
- Lorsqu'il s'agit de naviguer au travers des objets référencés par un attribut historique, opération qu'on appellera *navigation point-par-point*. Ceci est le cas par exemple, si l'on veut observer pour chaque employé, les superviseurs qu'il a eus au cours du temps.

Les deux requêtes suivantes illustrent chacun de ces cas.

**Q1 : Produit temporel**

*Pour chaque unité de production, donner l'historique de son volume de production pondéré par la qualité. On définit la production pondérée par la qualité comme étant le produit du volume de production par la qualité, pour un jour donné.*

```
/* type du résultat : struct<unité: UnitéDeProduction,
hist_prod: bag<vt: struct<binf: Date, bsup: Date>, vs: float>> */
select struct(unité: u,
              hist_prod: select struct(vt: struct(binf: max(set(iu.vt.binf, is.vt.binf)),
                                                  bsup: min(set(iu.vt.bsup, is.vt.bsup)) )
                                       vs : ip.vs * iq.vs)
              from u.hist_production as ip, u.hist_qualité as iq
              where not (iq.vt.bsup < ip.vt.binf or ip.vt.bsup < iq.vt.binf))
from LesUnités as u
```

**Q2 : Navigation point-par-point**

*Pour chaque ouvrier, donner l'historique de ses superviseurs.*

```
/* type du résultat : bag<struct<ouvrier: Ouvrier,
hist_superviseur : bag<struct<vt: struct<binf: Date, bsup: Date>, vs: Superviseur>>>> */
select struct(ouvrier : o,
              hist_superviseur : select struct(vt : struct(binf: max(set(iu.vt.binf, is.vt.binf)),
                                                           bsup: min(set(iu.vt.bsup, is.vt.bsup))),
                                               vs : is.vs))
                                 from o.hist_unité as iu, iu.vs.hist_superviseur as is
                                 where not (iu.vt.bsup < is.vt.binf or is.vt.bsup < iu.vt.binf)
from LesOuvriers as o
```

Ces deux requêtes sont formulées selon un schéma similaire. Prenons par exemple la deuxième d'entre elles. Le bloc select emboîté construit l'historique des superviseurs d'un employé donné. À cet effet un itérateur est introduit sur la suite d'affectations de chaque employé (variable iu). Pour chacune de ces affectations les instantanés de l'historique des superviseurs de l'unité référencée qui intersectent l'intervalle de l'affectation sont retrouvés (variable is). Pour chacun des instantanés ainsi sélectionnés un nouvel instantané est généré et inclus dans l'historique résultant. Noter que deux intervalles [bi1, bs1] et [bi2, bs2] s'intersectent si et seulement si on n'a pas bs1 < bi2 ni bs2 < bi1. Dans ce cas l'intersection est l'intervalle [max(bi1, bi2), min(bs1, bs2)]. On suppose que les opérateurs max et min d'OQL opèrent sur des collections de dates même si ce point n'est pas explicité dans la définition du standard.

Les historiques produits par les deux requêtes ci-dessus peuvent violer la contrainte de maximalité introduite dans le § 2.2.2. Dans le cas de la requête Q2 cette situation peut survenir si par exemple un ouvrier est muté d'une unité de production donnée vers une autre supervisée par le même superviseur. Par exemple considérons un ouvrier o tel que :

o.unité = { ... ⟨[1/2/98..2/5/98], u1⟩, ⟨[3/5/98..12/7/98], u2⟩ ... }
u1.superviseur = { ... ⟨[3/5/98..4/5/99], s ⟩ ... }
u2.superviseur = { ⟨[1/1/98..31/12/98], s⟩... }

La requête Q2 produit alors un n-uplet de la forme :

⟨o, { ... ⟨[1/2/98..2/5/98], s⟩, ⟨[3/5/98..12/7/98], s⟩ ... }⟩

qui contient un historique violant la contrainte de maximalité. Si l'on voulait assurer cette contrainte on obtiendrait une formulation de la requête sensiblement plus complexe.

Lorsque le nombre d'historiques intervenant dans une opération de mise en correspondance de valeurs synchrones augmente la complexité de l'expression des requêtes s'accroît considérablement comme l'illustre l'exemple suivant.

**Q3 : Navigation et comparaison point-par-point d'historiques**

*À quels instants, l'unité supervisée par l'employé de nom X a-t-elle eu une production, pondérée par la qualité, supérieure à celle supervisée par l'employé de nom Y?*

```
/* type du résultat: set<Date> */
select distinct pX.vt
from LesSuperviseurs as superX, LesSuperviseurs as superY,
     superX.hist_unité as iuX, superY.hist_unité as iuY,
     iuX.vs.hist_production as ipX, iuY.vs.hist_production as ipY,
     iuX.vs.hist_qualite as iqX, iuY.vs.hist_qualite as iqY
where superX.nom = "X" and superY.nom = "Y"
  and not (iuX.vt.bsup < iuY.vt.binf or iuY.vt.bsup < iuX.vt.binf)
  and (ipX.vt >= iuX.vt.binf and ipX.vt <= iuX.vt.bsup)
  and (ipY.vt >= iuY.vt.binf and ipY.vt <= iuY.vt.bsup)
  and ipX.vt = ipY.vt and iqX.vt = iqY.vt and iqX.vt = ipX.vt
  and ipX.vs * iqX.vs > ipY.vs * iqY.vs
```

Dans cette expression, il est intéressant d'observer la structure de la clause where, dans laquelle on voit successivement apparaître : une condition d'intersection entre intervalles, deux conditions d'inclusion d'un instant dans un intervalle, et trois égalités entre instants, et enfin, la condition sur la production pondérée. Ceci est dû au fait que cette requête combine des historiques représentés "par instants" avec d'autres représentés "par intervalles". Ce qui nous amène à la remarque suivante : si l'on venait à modifier la représentation utilisée pour un attribut historique, toutes les requêtes faisant intervenir cet attribut devraient être récrites. Or, comme nous le montrons dans [DFS00], le passage d'une représentation à une autre engendre des expressions de requêtes si différentes, qu'il est difficile d'utiliser l'expression dans l'ancienne représentation pour en dériver une dans la nouvelle.

Une autre source de complexité dans l'expression de requêtes temporelles, réside dans l'utilisation intensive que font celles-ci d'un large éventail d'opérations d'aggrégation sur des historiques. De telles requêtes sont particulièrement usuelles dans le contexte des applications financières. Ainsi, au moins 80% des requêtes apparaissant dans le banc d'essais pour systèmes de gestion de données temporelles pour les finances FinTime [JS99a], comportent une certaine forme d'agrégation temporelle. À ce propos, il convient de signaler que les données temporelles manipulées par ces applications sont souvent structurées en des *séries chronologiques*, c'est-à-dire des suites de valeurs estampillées par des instants pris dans un ensemble possédant une certaine périodicité [SDDM95, LEW96]. Ceci est le cas par exemple, de l'attribut hist_production de la classe Unité, en supposant que la production d'une unité est définie tous les jours de la semaine à l'exception des jours fériés.

Nous donnons ci-dessous des exemples types de requêtes comportant des agrégations

temporelles. Pour ne pas noyer la discussion dans des détails techniquesΓnous ne développons pas l'expression de chacune d'entre elles en OQL.

- "Pour chaque unité de productionΓdonner l'historique du nombre d'employés affectés à cette unité". Il s'agit ici d'appliquer un opérateur d'agrégation à chaque historique dans une collection.
- "Pour chaque unitéΓdonner l'historique de ses productions hebdomadaires". Cette requête correspond en fait à un *changement de granularité*.
- "Pour chaque jour de l'année 2000Γdonner le total des productions effectuées par l'unité U depuis le début de l'année". Il s'agit ici d'une *agrégation cumulative*.
- "Quelle est la moyenne des productions de l'unité supervisée par l'employé X par période de 10 jours consécutifs entre le 1/1/2000 et le 30/6/2000Γsachant que l'on considère des intervalles pas forcément disjoints : les intervalles [1/1/2000Γ 10/1/2000] et [2/1/2000Γ 11/1/2000] doivent tous les deux figurer dans le résultat". Ce type d'opération est communément appelée *moving window average* dans la terminologie anglosaxone.
- "Pour chaque journée où au moins une unité est en productionΓdonner la moyenne des facteurs qualité attribués aux unités en production à cette date". Ce type d'agrégationΓ qu'on appelera par la suite *agrégation transversale point par point*Γfait intervenir à chaque instantΓla valeur d'un attribut de type historique (ici l'attribut hist_production)Γ pour chacun des objets dans une collection (ici la collection des unités de production).

La complexité de la formulation de requêtes sur des données temporelles dans les langages d'interrogation classiques a été étudiée sous différents angles dans un bon nombre de travaux. À titre d'exemple nous citons [Can97]Γqui dresse une liste de requêtes temporellesΓstructurée autour d'une typologieΓet considère pour certaines leur expression en OQL. Dans le cadre des modèles relationnelsΓle travail le plus complet en la matière est sans doute [Sno99]Γ qui fournit une excellente analyse d'un grand nombre de requêtes temporellesΓet étudie pour chacune plusieurs formulations alternatives en SQL2. Ce travail aborde également l'expression de requêtes de mise à jour et de contraintes d'intégrité. Bien que moins exhaustifΓ[Sno95b] présente aussi quelques requêtes temporelles complexesΓavec leur expression en SQL2.

Beaucoup de travaux se sont aussi penchés sur l'expression et l'évaluation de requêtes sur des séries chronologiques [Sha99]. Ces travaux montrent en particulier qu'une grande partie des requêtes sur ce type de données sortent du cadre du pouvoir d'expression de SQL et d'OQL "pur" (c.à.d. sans appel de méthodes)Γcar elles font intervenir des fonctions d'agrégation variées et complexes. Cet état de fait a d'ailleurs motivé les éditeurs de SGBD relationnels et "relationnels-objet" à fournir des modules spécifiques pour la manipulation de ces types de données (voir par exemple Oracle [Ora97] et Informix [Inf97]). Dans le contexte des SGBD à objetsΓde telles extensions ne sont pas toujours nécessaires : le mécanisme d'appel de méthodes fournissant un moyen satisfaisant de prendre en compte ces besoins.

Une autre catégorie de requêtes très récurrentes dans les applications manipulant des séries chronologiques concerne la recherche de *motifs* d'évolution. Parmi les requêtes représentatives de cette catégorie on peut citer celles où il s'agit de mettre en évidence des sous-séquences communes à plusieurs séries (requêtes dites de *subsequence matching* ou encore de *similarity search* dans la terminologie anglosaxone [FRM94]) comme par exemple dans *"Quelles unités ont une courbe de production similaire à celle d'une unité donnée ?"*.

## 2.3 Bases de Données Temporelles

Face à la complexité engendrée par la manipulation d'associations temporelles aussi bien au niveau de la conception que de l'implantation d'applications de nombreux travaux de recherche dans le domaine des Bases de Données ont été entrepris. Regroupés dans une branche connue sous le nom de Bases de Données Temporelles ces travaux visent de façon générale à fournir des abstractions permettant aux concepteurs et développeurs de maîtriser cette complexité en s'appuyant sur des méthodologies et/ou des outils spécifiques.

Au niveau de la conception de telles abstractions ont donnée lieu à des extensions de modèles entité-association (voir [GJ99] pour une synthèse) et à objets (p. ex. [PSZ+97]). Au niveau de la modélisation logique et de l'expression de requêtes de nombreuses extensions de modèles relationnels et à objets ont également été proposées (voir plus loin). Enfin au niveau physique des structures de données et des algorithmes spécifiques à la gestion de relations temporelles ont été étudiés [ST99 Zur97]. Conformément au cadre de cette thèse nous nous concentrons par la suite sur les travaux se situant au niveau de la couche logique.

### 2.3.1 Extensions temporelles de modèles relationnels classiques

La littérature sur les bases de données temporelles abonde en propositions d'extensions temporelles de modèles de données et de langages de requêtes comme l'attestent les compilations successives de références bibliographiques sur le domaine dont la plus récente [WJSW98]. En particulier on décompte une trentaine d'extensions temporelles du modèle relationnel et une quinzaine concernant des modèles à objets. Les références [TCG+93] [CT95] et [EJS98] contiennent des descriptions détaillées de certaines de ces propositions.

Face à cette diversité des efforts d'unification ont été entrepris surtout dans le cadre du modèle relationnel en première forme normale. Ces efforts ont abouti entre autres à un glossaire de concepts [JD98] et au langage TSQL2 défini après consensus d'un grand nombre de chercheurs dans le domaine. Une variante de ce dernier langage baptisée ATSQL [SBJS98] a fait l'objet d'une proposition aux comités de standardisation ANSI et ISO/SQL3.

D'autres travaux de synthèse dans ce domaine ont été entrepris sous forme d'états de

l'art et d'études comparatives. Par exemple [MS91] analyse un certain nombre d'extensions temporelles de l'algèbre relationnelle et compare leur pouvoir d'expression. Toujours dans cette perspective d'étude de l'expressivité [CCT94] et [TT98] comparent des extensions temporelles de modèles de données relationnels classiques vis-à-vis des extensions de modèles non en première forme normale (N1NF) et établissent des théorèmes d'équivalence entre des algèbres et des calculs sur ces modèles. [Cho94] quant à lui offre un excellent état de l'art sur les langages de requêtes relationnels temporels avec un certain effort d'unification et une ouverture vers des considérations théoriques (formalisation pouvoir d'expression complexité d'évaluation). Enfin [ZCF+97 JS99b] fournissent des synthèses relativement actualisées sur les modèles et langages pour données temporelles avec un accent sur les modèles relationnels.

Parmi les propositions de modèles et de langages relationnels pour données temporelles nous n'en détaillons que deux par la suite. Premièrement TSQL2 dans la mesure où il est la synthèse d'un grand nombre de résultats de recherche sur les bases de données temporelles et qu'il possède une documentation assez complète. Ensuite SQL/TP car il s'agit du seul modèle utilisant exclusivement des estampilles temporelles de type instant pour lequel un modèle d'évaluation "réaliste" a été proposé. Certes des travaux tels que [Lor93 Dar98 CZ99] ont mis en évidence les avantages d'une représentation "par instants" des relations temporelles mais ils n'ont pas étudié des techniques permettant d'évaluer les requêtes sur ces représentations avec une complexité algorithmique comparable à celle qu'on obtiendrait si on appliquait des techniques d'évaluation classiques (c'est-à-dire celles associées à l'algèbre relationnelle) sur une représentation "par intervalles". En conséquence les requêtes temporelles exprimées dans les langages proposés par ces travaux s'évaluent de façon sensiblement moins efficace que si elles étaient exprimées directement en SQL ce qui est contraire aux objectifs de performances recherchés dans les SGBD.

### TSQL2

Du point de vue du système de types TSQL2 étend SQL2 par le type Period modélisant des ensembles finis d'instants convexes. Ce type temporel vient s'ajouter à ceux déjà présents dans SQL2 à savoir Date Timestamp Interval et Time. À ce propos on rappelle que les types Date et Timestamp modélisent des instants à la granularité du jour et de la seconde respectivement le type Interval modélise des durées signées à n'importe quelle granularité du calendrier grégorien et le type Time modélise des durées positives à la granularité de la seconde de taille comprise entre 0 et 24 heures [8]. Dans tous les cas ces types temporels sont donc définis par rapport à des granularité fixées appartenant au calendrier grégorien. Pour combler cette limitation TSQL2 fournit des mécanismes permettant de spécifier des instants

8. Les valeurs de type Timestamp et Time sont définies par rapport à une localisation géographique, ce qui permet de tenir compte des décalages horaires.

durées et périodes dans des granularités propres à chaque application (p. ex. des trimestres), éventuellement définies à partir de calendriers autres que le Grégorien (p. ex. lunaire).

Le type Temporal Element modélisant des ensembles finis d'instants non forcément convexes est aussi implicitement présent dans le modèle de données de TSQL2, même si aucun opérateur permettant de manipuler directement des valeurs de ce type n'est fourni.

En TSQL2, les entités et associations temporelles sont représentées sous forme de n-uplets possédant un attribut implicite de type temporel. De tels n-uplets sont regroupés dans des *relations temporelles*, qui peuvent être soit de type *événement*, soit de type *état*, suivant que l'attribut temporel est de type instant ou ensemble d'instants.

Suivant une dichotomie introduite dans [SA85] et devenue incontournable dans le domaine des Bases de Données Temporelles, TSQL2 distingue le *temps de validité* du *temps de transaction*. Le temps de validité d'un fait, correspond aux instants auxquels ce fait est vrai dans le monde modélisé, alors que le temps de transaction correspond aux instants où le fait est enregistré dans la base de données. Dans le cas général, ces deux notions sont orthogonales : un fait peut être observé dans le monde modélisé à une date donnée (p. ex. le 30/4/99), mais n'être enregistré que quelque temps plus tard (p. ex. le 3/5/99). Réciproquement, dans les applications où l'on peut connaître à l'avance l'occurrence d'un fait (*tel employé va être embauché à partir de telle date*), il est possible que des faits soient enregistrés avant d'être observés dans le monde modélisé[9].

Afin d'intégrer la dichotomie temps de transaction vs. temps de validité, les relations temporelles en TSQL2 sont classifiées en relations *historiques* et relations *de reprise*. Dans les premières, l'attribut temporel porte la sémantique du temps de validité. Sa valeur est fournie par l'utilisateur au moment de la saisie. Les relations de reprise quant à elles, correspondent au temps de transaction. La valeur de l'attribut temporel est fixé automatiquement par le système lors de chaque insertion ou suppression de n-uplets (non-estampillés) dans la relation. Enfin, il est possible en TSQL2, comme d'ailleurs dans beaucoup de ses prédécesseurs et successeurs, d'observer l'évolution d'une relation à la fois suivant le temps de validité et de transaction. Ces relations sont alors qualifiées de *bitemporelles*. Elles comportent non pas un seul, mais deux attributs temporels, dont un est entièrement contrôlé par l'utilisateur, alors que l'autre est fixé par le système lors des mises à jour.

La table 2.1 décrit un exemple de relation bitemporelle en TSQL2. L'attribut libellé Valid (resp. Transaction) dénote le temps de validité (resp. de transaction). À titre d'exemple, le premier n-uplet se lit comme suit : entre le 1/12/98 et le 2/4/99 (exclu), il est enregistré qu'entre le 1/12/98 et le 1/2/99, l'employé de nom Dumas gagne un salaire de 2000. Le

9. Il existe des cas où la distinction entre le temps de validité et le temps de transaction n'est pas pertinente. Ceci est le cas par exemple lorsque la base de données reçoit des informations en temps réel à partir de capteurs. [JS94] parle de *base de données bitemporelle dégénérée* pour référencer cette situation.

deuxième n-uplet établit que cette connaissance a été "corrigée" le 2/4/99Γdate à laquelle il a été enregistré qu'entre le 1/12/98 et le 1/2/99Γl'employé Dumas gagnait 2100 et non pas 2000. Le symbole UC (*Until Changed*) figurant dans ce deuxième n-upletΓindique que l'intervalle de transaction de ce n-uplet s'étale jusqu'à la date d'aujourd'hui. Chaque fois que la relation est accédée au travers d'une requêteΓce symbole est remplacé par la date indiquée par l'horloge du système [CDI+97].

| Nom | Salaire | Valid | Transaction |
|---|---|---|---|
| Dumas | 2000 | { [1/12/98..1/2/99) } | [1/12/98..2/4/99) |
| Dumas | 2100 | { [1/12/98..1/2/99) } | [2/4/99..UC) |
| Dumas | 2500 | { [1/2/99..1/4/99) } | [3/2/99..UC) |
| Dumas | 2600 | { [1/4/99..31/7/99)Γ [1/9/99..31/12/99)} | [2/4/99..UC) |

TAB. 2.1 – *Exemple de relation bitemporelle en TSQL2.*

Dans le but de simplifier l'exposéΓnous nous limitons dans la suite à la manipulation du temps de validité. Nous reviendrons sur le concept de temps de transaction en temps utile.

Voici une description du schéma de la base de données "Usine" en TSQL2.

```
create table LesEmployés(nom varchar(20), salaire number(6,2),
                         primary key (nom))
                         as valid state Day
/* le mot-clé state indique que les estampilles sont de type "ensemble d'instants". */
create table LesProductions(codeUnité varchar(10), production number(8), qualité number(1, 4)
                            primary key (codeUnité))
                            as valid event Day
/* le mot-clé event" indique que les estampilles sont des instants. */
create table LesSuperviseurs(codeUnité varchar(10), nom varchar(20),
                             primary key(codeUnité),
                             foreign key (nom) references LesEmployés(nom))
                             as valid state Day
create table LesOuvriers(codeUnité varchar(10), nom varchar(20),
                         foreign key (nom) references LesEmployés(nom))
                         as valid state Day
```

Dans ce schémaΓles contraintes d'intégrité de clé primaire et étangère ont une sémantique "temporelle". AinsiΓla contrainte établissant que nom est une clé de LesEmployésΓn'entraîne pas que deux n-uplets de cette relation ont forcément des valeurs différentes pour cet attributΓ mais plutôtΓque le cas échéant les deux n-uplets concernés ont des estampilles temporelles disjointes. En d'autres termesΓcette contrainte modélise le fait qu'un employé ne possède qu'un seul salaire à une date donnéeΓmême si ce salaire peut varier au cours du temps.

Nous donnons ci-dessous l'expression en TSQL2 de la requête Q2. Nous invitons le lecteur à comparer cette expression avec celle en OQL qui figure page 25.

**Q2** *(reprise)* : **Produit naturel temporel**

*Pour chaque ouvrier, donner l'historique de ses superviseurs.*

```
select o.nom, s.nom
from LesOuvriers as o, LesSuperviseurs as s
where o.codeUnité = s.codeUnité
```

En l'absence de prédicat temporel dans la clause where, la sémantique du produit cartésien des deux relations de la clause from est étendue. Plus précisément, l'estampille temporelle de chaque n-uplet résultat est l'intersection des estampilles des deux n-uplets arguments. Tout se passe alors comme si l'expression de la requête ci-dessus (qui ne fait aucune référence explicite au temps) était évaluée sur chaque état successif de la base, et que les résultats de chacune de ces évaluations étaient ensuite regroupés en une seule relation temporelle.

Dans ATSQL [SBJS98], la variante de TSQL2 proposée aux comités de standardisation ANSI et ISO/SQL3, ce principe d'évaluation *point par point* est généralisé à tous les opérateurs de SQL2 (y compris ceux de mise à jour). Des techniques efficaces pour l'évaluation de requêtes temporelles ainsi exprimées ont été développées et implantées dans deux systèmes : le prototype TIGER [10] et le système TimeDB [11] (qui est en passe de devenir un produit commercial). Plus généralement, [Böh95] dresse une liste de prototypes de SGBD temporels, et fournit des indications sur leurs caractéristiques techniques et leur disponibilité.

### SQL/TP

Par rapport aux autres propositions d'extension temporelle de SQL2, l'originalité de SQL/TP [Tom98] est de distinguer deux niveaux de modélisation d'une base de données temporelles : le *niveau abstrait* sur lequel sont basées la syntaxe et la sémantique du langage de requêtes, et le *niveau concret* sur lequel repose le modèle d'évaluation de ce langage. Ainsi, SQL/TP permet une modélisation point par point de l'évolution d'une relation (au niveau abstrait les n-uplets sont toujours estampillées par des instants), tout en fournissant un modèle d'évaluation réaliste (au niveau concret une représentation par intervalles est possible).

Nous décrivons ci-dessous le schéma de notre base de données exemple en SQL/TP. Les contraintes de clés primaires et étrangères qui y figurent ont la même sémantique qu'en SQL2, mais elles sont définies par rapport à la vue abstraite de la base de données.

```
create table LesEmployés (nom varchar(20), salaire number(6, 2),
```

10. http://www.cs.auc.dk/~boehlen
11. http://www.timeconsult.com

```
                    ts time using unbounded intervals,
                    primary key (nom, ts))
/* Au niveau abstrait, chaque n-uplet de cette relation est estampillé par un attribut ts de type
instant. Au niveau concret, l'attribut temporel est de type "intervalle non borné"; les constantes
−∞ et +∞ peuvent constituer l'une, ou les deux bornes de ces intervalles. */
create table LesProductions (codeUnité varchar(20), production number(8), qualité number(1, 4),
                    ts time using points,
                    primary key (codeUnité, ts))
/* La relation est représentée par instants aussi bien au niveau abstrait que concret. */
create table LesSuperviseurs(codeUnité varchar (10), nom varchar (20),
                    ts time using unbounded intervals,
                    primary key (codeUnité, ts),
                    foreign key (nom, ts) references LesEmployés(nom, ts))
create table LesOuvriers(codeUnité varchar (10), nom varchar (20),
                    ts time using unbounded intervals
                    foreign key (nom, ts) references LesEmployés(nom, ts))
```

Sur ce schémaΓla requête Q2 s'écrit :

**Q2** *(reprise)* : **Navigation temporelle point-par-point.**

*Pour chaque ouvrier, donner l'historique de ses superviseurs.*

```
select o.nom, s.nom
from LesOuvriers as o, LesSuperviseurs as s
where o.codeUnité = s.codeUnité and o.ts = s.ts
```

Notons que contrairement à TSQL2Γen SQL/TP la condition de jointure sur le temps doit être explicitée (au moyen de la condition o.ts = s.ts dans la clause where). À première vueΓceci peut paraître un désavantage de SQL/TP par rapport à TSQL2. CependantΓcette démarche permet de gagner en uniformité. Pour illustrer ce pointΓconsidérons une requête d'agrégation transversale point-par-pointΓconcept introduit dans le § 2.2.3 page 33.

**Q4 : Agrégation transversale point-par-point**

*À chaque instant, combien d'employés perçoivent un salaire horaire supérieur à 100.*

| TSQL2 | SQL/TP |
|---|---|
| select Valid (E), count(nom) | select ts, count(nom) |
| from LesEmployes as E | from LesEmployés |
| where salaire > 100 | where salaire > 100 |
| group by Valid (E) using Instant | group by ts |

La clause group by Valid (E) using Instant dans l'expression en TSQL2Γspécifie que la relation LesEmployés (restreinte aux n-uplets satisfaisant le prédicat de la clause where) est

regroupée instant par instant. Plus précisément, un groupe est généré pour chaque instant référencé par au moins un n-uplet dans la relation LesEmployés une fois restreinte. Chacun de ces groupes contient tous les n-uplets de cette relation dont l'attribut temporel contient l'instant définissant le groupe. L'expression Valid(E) dans la clause select indique que la relation résultante possède un attribut temporel, qui est du même type que l'attribut temporel de la relation E. En l'occurrence, la relation résultante est de type état et son attribut temporel est donc de type "ensemble d'instants". De ce fait, tous les groupes pour lesquels le résultat de l'agrégation est le même, sont regroupés en un seul n-uplet dans le résultat.

En fait, la syntaxe et sémantique de la clause group by en TSQL2 est beaucoup plus complexe que ne le laisse entrevoir l'exemple ci-dessus. Pour preuve, un chapitre entier de [Sno95b] est consacré à ce point particulier. Ceci contraste avec la simplicité de SQL/TP, qui atteint le même pouvoir d'expression que TSQL2, sans modifier ni la syntaxe ni la sémantique de la clause group by de SQL2, si ce n'est que la requête opère (du moins conceptuellement) sur la représentation abstraite.

Avant de conclure, on remarque que la formulation de la requête ci-dessus en SQL2 non-étendu (en estampillant les n-uplets par des intervalles) est extrêmement complexe. À notre connaissance, les formulations les plus concises de ce type de requêtes font de l'ordre de 30 lignes de code. Cette complexité provient de l'inadéquation de la représentation par intervalles lorsqu'on raisonne par instants. Plus généralement, la complexité de l'expression de requêtes temporelles provient souvent, non pas de la requête en elle-même, mais du fait que le langage utilisé impose de manipuler une représentation des associations temporelles inadéquate vis-à-vis de cette requête.

### 2.3.2 Extensions temporelles de modèles relationnels N1NF

Le point commun des extensions temporelles de modèles de données relationnels N1NF réside dans l'utilisation du concept d'*historique*[12] : fonction à domaine dans un ensemble d'instants fini [CC93, GN93, SS93, SLR96] ou infini [EGMS99].

Selon les modèles, les valeurs de type historique ont une existence par elles-mêmes, ou seulement lorsqu'elles sont attachées à un n-uplet d'une relation temporelle par le biais d'un de ses attributs. Ainsi par exemple, HRDM [CC93] et TempSQL [GN93] ne comportent pas d'opérateurs sur les historiques à proprement parler, mais sur des relations comportant des attributs de ce type. De même, dans [SS93] les opérateurs d'interrogation n'opèrent pas directement sur des historiques, mais sur des collections d'historiques regroupés au sein de relations de la forme (S, T, A), où S est un attribut dénotant un identificateur d'entité (p.

12. La dénomination exacte de ce concept varie suivant les auteurs : *time function* [CC93], *temporal assignment* [GN93], *time sequence* [SS93, SLR96], ou encore *moving object* [EGMS99].

ex. le code d'une unité de production)Γ T est un attribut de type instant ou intervalleΓet A est l'attribut dénotant les valeurs prises par le phénomène observé (p. ex. le volume de production des unités).

[SLR96] et [EGMS99] par contreΓtraitent les historiques comme des types abstraits de données (TAD)Γc'est-à-dire qu'ils définissent des constructeursΓdes sélecteurs et des opérateurs algébriquesΓpermettant de manipuler les valeurs de type historique indépendamment de leurs représentations. Les auteurs étudient ensuite des implantations des opérateurs fournis par leurs modèles respectifsΓen fixant différents modes de repésentation. À ce proposΓ [SLR96] se restreint aux représentations à base d'instantsΓpuisque le spectre des applications visées est celui des séries temporelles pour les finances. De mêmeΓ[EGMS99]Γet plus récemment [FGNS00]Γconsidèrent des représentations à base d'objets géométriquesΓpuisqu'orientés vers la manipulation de trajectoires (mouvements de points).

De façon généraleΓles extensions temporelles de modèles relationnels à base de TAD se révèlent particulièrement adaptées aux applications financières manipulant des séries chronologiques (Cf. § 2.2.3). D'ailleursΓles modules dédiés aux séries chronologiques offerts par des SGBD commerciaux tels qu'Oracle [Ora97] et Informix [Inf97]Γse basent sur ce type d'approche. ToutefoisΓcomme nous le montrons dans [DFS00]Γle langage résultant de l'intégration dans SQL des opérateurs sur les TAD définis par le module "séries chronologiques" d'Oracle (qui est très proche de celui d'Informix) est assez intriquéΓpuisqu'il résulte du mélange d'un paradigme "logique" (SQL) avec un paradigme "fonctionnel" (l'approche TAD). En ce sensΓnous pensons que les approches de manipulation de données temporelles à base de TAD sont mieux adaptées à des modèles à objetsΓqu'à des modèles relationnels. C'est en partie sur cette idée que se base le travail que nous développons dans les chapitres ultérieurs.

### 2.3.3 Extensions temporelles de modèles à objets et modèles à versions

HistoriquementΓles travaux autour des modèles de données temporelles à objets se divisent en deux périodes bien différenciées : la période "avant ODMG" et la période "après".

La première période se caractérise par un manque de consensus autour des concepts et fonctionnalités attendues dans un SGBD à objets. AinsiΓles travaux développés au cours de cette période reposent sur des modèles de données hétéroclitesΓcertains démunis d'une sémantique formelle ou même d'une implantation. Parmi ces travaux nous distinguons :

- [WD93] qui étudie la manipulation d'historiques au dessus de OODAPLEXΓune extension orienté-objet de DAPLEX [Shi81]. L'un des apports majeurs de ce travail a été de promouvoir l'utilisation exclusive d'estampilles de type instant pour représenter des associations temporellesΓidée reprise par la suite dans SQL/TP (voir ci-dessus).

Malheureusement, les auteurs n'ont pas abordé les aspects "évaluation de requêtes", qui constitue sans doute l'un des défis majeurs posés par cette approche.

- [RS91] et [RS93] qui présentent respectivement un modèle de données et un langage de requêtes temporels, fondés sur un modèle à objets *ad hoc* défini par les mêmes auteurs.
- [GO93] qui introduit des types temporels dans un modèle de données "tout-objet" baptisé TIGUKAT [OPS+95]. Les idées développées lors de la conception de ces types ont été reprises et étendues par la suite pour donner lieu à un *framework* à objets pour la gestion de données temporelles [GOS98]. Ce *framework* prend en compte une multitude d'aspects, comme par exemple la structure du domaine temporel (continu ou discret, linéaire ou arborescent), la manipulation de valeurs temporelles exprimées à différents niveaux de granularité et selon des multiples calendriers, et la modélisation d'historiques. À ce titre, il représente un effort d'intégration de travaux existants.

D'autres travaux de cette "première vague" que nous ne décrivons pas ici, incluent [SC91] [EPP93] et [BFG96]. Le lecteur intéressé trouvera dans [Sno95a] une étude comparative des propositions issues de certains de ces travaux[13]. Dans cette étude, l'auteur développe une critique envers l'idée d'étendre des modèles à objets par des fonctionnalités temporelles. Cette critique est fondée sur l'observation que ces modèles sont intrinsèquement extensibles, et que cette extensibilité suffit à intégrer la dimension temporelle de manière satisfaisante, sans avoir à modifier les fondements du modèle. Cette affirmation est plus ou moins fausse selon le modèle à objets considéré. Nous pensons en particulier qu'elle ne s'applique pas au modèle de l'ODMG, dans lequel les mécanismes d'extensibilité sont très limités. Ainsi, il n'est pas possible dans ce modèle de définir des classes paramétrées, ce qui permettrait d'encapsuler de façon générique la notion d'historique. En fait, comme nous le verrons par la suite, même si l'on venait à encapsuler le type historique dans une classe paramétrée, la manipulation des valeurs de ce type nécessiterait l'utilisation d'opérateurs d'ordre supérieur (c.à.d. des opérateurs prenant des fonctions en paramètre), dont la spécification n'est pas supportée par la plupart des modèles à objets existants.

L'émergence du standard ODMG en 1993-94 a donné lieu à une deuxième vague de travaux autour des modèles de données temporelles à objets. Parmi les travaux de cette "deuxième vague" (dans lesquels s'inscrit TEMPOS), on distingue TAU [KT96], T_ODMG [BFGM98] et TOOBIS [TOO97][14], que nous décrirons par la suite.

Parallèlement aux travaux référencés ci-dessus, d'autres travaux plus ou moins similaires concernant l'intégration du concept de *version* dans les bases de données à objets ont été menés [CK86, Sci91, Sci94, GJ94, AHP97, AJ97]. Les modèles issus de ces travaux permettent de décrire des états successifs et/ou alternatifs d'un ensemble d'objets, ce qui les rend a priori

13. Un autre état de l'art assez exhaustif sur les travaux de cette période est fourni dans [Skj97]
14. L'extension d'OQL esquissée dans [FE98] appartient aussi aux travaux de cette période.

des bons candidats pour répondre aux besoins des applications temporelles. Un examen plus détaillé montre cependant que vis-à-vis de ces besoins les modèles à versions possèdent de nombreux défauts. Nous reviendrons sur ce point à la fin de cette section.

### Extensions temporelles de l'ODMG: le projet TOOBIS

Alors que les travaux autour de TOOBIS couvrent une large partie des composantes de l'ODMG (le modèle de données ODL OQL et la passerelle C++) les autres extensions temporelles de l'ODMG à savoir TAU et T_ODMG se concentrent sur le modèle de données. Par ailleurs à quelques détails près ces trois propositions gravitent toutes autour des mêmes concepts de base. Aussi nous nous limitons par la suite à présenter TOOBIS.

Le projet européen TOOBIS (1996-97) [TOO97 SSV98] avait pour but entre autres de proposer une extension temporelle du standard de l'ODMG. Dans cette optique les principaux résultats du projet concernent la spécification d'une extension du modèle d'objets de l'ODMG (baptisé TODM [TOO96a]) ainsi que des extensions temporelles des langages ODL et OQL (resp. TODL et TOQL [TOO96b]).

TODM étend le modèle d'objets de l'ODMG en y introduisant des types pour la manipulation de valeurs temporelles complexes (intervalle ensemble d'instants) observées à différents niveaux de granularité. Il introduit aussi les notions de classe temporelle et de propriété temporelle. Les classes temporelles fournissent un mécanisme d'historisation à base de versions d'objets : une instance d'une classe temporelle est composée d'une collection de versions estampillées par des instants ou par des intervalles (voir ci-dessous). Les propriétés temporelles quant à elles constituent des mécanismes d'historisation des valeurs successives prises par une propriété relativement à chacun des objets d'une classe : une *propriété temporelle* possède un historique c'est-à-dire une collection de valeurs estampillées soit par des instants soit par des intervalles. Comme dans TSQL2 le type des estampilles (instant ou intervalle) est déterminé par le type de la propriété qui peut être soit d'état soit d'événement. Dans tous les cas il est possible d'observer l'évolution des données suivant le temps de transaction suivant le temps de validité ou suivant les deux à la fois.

TOOBIS étend la notion d'intégrité référentielle définie par l'ODMG aux associations temporelles. Par rapport à notre base de données exemple ce concept d'*intégrité référentielle temporelle* permet d'exprimer au niveau de la définition du schéma la contrainte *"à tout instant, le superviseur d'une unité supervise cette unité"* (Cf. § 2.2.2). Nous donnons ci-dessous la spécification du schéma de la base de données exemple en TODL :

```
class UnitéDeProduction(extent LesUnités, key codeUnité) {
    attribute string codeUnité;
    valid state granularity day relationship Superviseur superviseur inverse Superviseur::supervise;
```

```
    /* superviseur est une propriété temporelle (temps de validité) de type état et de granularité
    jour ; sa valeur historique est une collection de couples ⟨ Intervalle, Superviseur ⟩; par ailleurs,
    superviseur est l'un des chemins de traversée d'une association temporelle entre la classe
    UnitéDeProduction et la classe Employé dont le chemin inverse est la propriété supervise */
    valid state granularity day relationship set<Ouvrier> ouvriers inverse Ouvrier::unité;
    valid event granularity day attribute short production;
    /* production est une propriété temporelle de type événement; son historique est donc une
    collection de couples ⟨ Instant, entier ⟩ */
    valid event granularity day attribute float qualité;
}
class Employé (extent LesEmployés, key nom) {
    attribute string nom;
    valid state granularity day attribute float salaire;
}
class Ouvrier extends Employé (extent LesOuvriers) {
    valid state granularity day relationship UnitéDeProduction unité inverse UnitéDeProduction::unité;
}
class Superviseur extends Employé (extent LesSuperviseurs) {
    valid state granularity day relationship UnitéDeProduction supervise
                                        inverse Superviseur::superviseur;
}
```

Le langage de requêtes TOQL étend OQL en y introduisant les types temporels définis par TODM ainsi que les opérateurs associés. Les requêtes portant sur des classes et/ou des propriétés temporellesΓs'expriment principalement en raisonnant sur la représentation de leurs instances sous forme de collections de versions d'objets estampillées (dans le cas de classes) ou des collections de valeurs estampillées (pour les propriétés) (Cf. § 2.2.3). De ce faitΓles expression des requêtes en TOQL sont souvent très similaires à celles en OQLΓ modulo le fait que le type intervalle est encapsulé dans une interface prédéfinie. Pour illustrer ce pointΓnous écrivons ci-dessous la requête Q2 en TOQL. Le lecteur est invité à comparer cette expression avec celle en OQL donnée dans la page 25.

**Q2** *(reprise)* : **Navigation temporelle point-par-point**

*Pour chaque ouvrier, donner l'historique de ses superviseurs.*

```
/* Type du résultat: bag<struct<ouvrier: Ouvrier, hist_superviseur:
bag<struct<value: Superviseur, VT: Period>>>> */
select struct(ouvrier : o,
              hist_superviseur : select struct(value: s, VT: intersection(valid(u), valid(s))) )
                                 from valid o.unité as u, valid u.superviseur as s
                                 where valid(u) overlaps valid(s)
```

```
        /* le prédicat overlaps teste l'intersection de deux intervalles */
from LesOuvriers as o
```

En TOQL, le mot-clé valid possède plusieurs sémantiques selon le contexte. Tout d'abord, il permet d'obtenir l'historique d'une propriété temporelle. Ainsi, l'expression o.unité ci-dessus est précédée du mot-clé valid. Le cas échéant, cette expression serait de type UnitéDeProduction et serait évaluée par rapport à l'état "courant" de la base. D'autre part, le mot-clé valid permet d'accéder à la valeur temporelle d'un élément d'un historique (c'est le cas de valid(u) ci-dessus). Dans d'autres contextes, valid est utilisé pour construire des éléments d'historiques à partir d'une valeur et d'un intervalle.

Nous pensons que même si TOQL fournit un niveau d'abstraction supérieur à celui d'OQL lorsqu'il s'agit de formuler des requêtes temporelles, il garde néanmoins certains de ses défauts. En particulier, l'utilisateur est obligé de raisonner sur une représentation des historiques qui n'est pas toujours la plus adéquate vis-à-vis de son besoin. Par exemple, l'expression de la requête ci-dessus ne fait pas du tout ressortir le fait qu'on exprime une opération de navigation (on veut connaître le superviseur de l'unité de production d'un ouvrier). Tout comme en OQL, cela donne lieu à des expressions de requêtes assez complexes, lorsque le nombre de propriétés temporelles mises en jeu augmente. Pour donner une idée de cette complexité, nous reprenons la requête Q3 page 26 en TOQL.

**Q3** *(reprise)* : **Navigation et comparaison point-par-point d'historiques**
*À quels instants, l'unité supervisée par l'employé de nom X a-t-elle eu une production, pondérée par la qualité, supérieure à celle supervisée par l'employé de nom Y?*

```
/* type du résultat: set<Timestamp> */
select distinct valid(pX)
from LesSuperviseurs as superX, LesSuperviseurs as superY,
     valid superX.unité as iuX, valid superY.unité as iuY,
     valid iuX.vs.production as ipX, valid iuY.production as ipY,
     valid iuX.vs.qualite as iqX, valid iuY.qualite as iqY
where superX.nom = "X" and superY.nom = "Y"
  and valid(iuX) overlaps valid(iuY)
  and valid(ipX) >= begin(valid(iuX)) and valid(ipX) < end(valid(iuX))
  /* begin(X) (resp. end(X)) retrouve la borne inférieure (resp. supérieure) de intervalle X. */
  and valid(ipY) >= begin(valid(iuY.vt)) and valid(ipY) < end(valid(iuY))
  and valid(ipX) = valid(ipY) and valid(iqX) = valid(iqY) and valid(iqX) = valid(ipX)
  and ipX * iqX > ipY * iqY
```

### Modèles à base de versions

Dans les applications de conception et d'ingéniérie (génie logicielΓCAOΓédition de documentsΓetc.)Γla prise en compte de la nature *essai-erreur* des processus se révèle centrale dans la gestion des données. AussiΓles plate-formes dédiées à ces applications (p. ex. l'atelier de génie logiciel ADELE [EC94]) intègrent des concepts telles que *version*Γ*contexte* et/ou *configuration*Γpermettant de modéliser des états successifs et/ou alternatifs des objets et des associations manipulés. Dans le but de fournir un support à ces applications au niveau du SGBDΓde nombreux modèles de données à base de *versions d'objets* ont été proposés [CK86ΓSci91ΓSci94ΓGJ94ΓAHP97ΓAJ97]. Certaines de ces propositions ont d'ailleurs été intégrés à divers degrés dans des SGBD commerciaux tels que Versant [15]ΓObjectivity/DB [16]Γ et $O_2$

Dans les modèles issus de ces travauxΓun objet est vu comme un arbre dont les noeuds correspondent à des versionsΓet les arêtes dénotent des liens dits de *dérivation* entre ces versions. Le concept de propriété s'applique non plus aux objets en eux-mêmes comme c'est le cas dans les modèles à objets classiquesΓmais à chacune de leurs versions prises séparément. Autrement ditΓchaque version d'un objet dénote un état.

Il est naturel de penser que l'on peut modéliser les états pris par un objet au cours du temps au travers de versions liées par un arbre de dérivation entièrement unaire (c.à.d. une suite). ToutefoisΓmême si cette modélisation prend en compte le caractère "séquentiel" du tempsΓelle ne capture pas sa "métrique". Autrement ditΓcette modélisation ne décrit ni la durée de validité de chaque état d'un objetΓni la distance entre deux étatsΓni les relations temporelles entre des états pris par deux objets distincts. En conséquenceΓces aspects doivent être modélisés à l'intérieur de chaque versionΓpar exemple au travers d'estampilles temporelles. OrΓconsidérer qu'un objet temporel est une suite de versions estampillées par leur période de validité (comme le propose [Sci91])Γrevient conceptuellement à le modéliser comme une suite de n-uplets estampillés par des intervalles. Un objet de la classe Ouvrier est alors représenté de la façon suivante :

⟨ E1, { ⟨ V1, ⟨ binf: 1/1/98, bsup: 15/4/98, nom: "Daassi", salaire: 5000.0, affectation: U1 ⟩ ⟩,
⟨ V2, ⟨ date: 16/4/98, bsup: 15/11/98, nom: "Daassi", salaire: 5500.0, affectation: U1 ⟩ ⟩,
⟨ V3, ⟨ date: 1/1/99, bsup: 5/8/99, nom: "Daassi", salaire: 5500.0, affectation: U2 ⟩ ⟩ }

Où E1 dénote un identificateur d'objet de la classe OuvrierΓV1ΓV2 et V3 des identificateurs de versionΓet U1ΓU2 des identificateurs d'objets de la classe UniteDeProduction.

Cette représentation est une variante de la représentation par attributs de type "liste de couples ⟨intervalleΓvaleur⟩"Γdont nous avons étudié les inconvénients dans la section 2.2. De

15. http://www.versant.com
16. http://www.objectivity.com

plusΓle fait que les estampilles temporelles sont attachées aux états des objets et non pas aux valeurs prises par leurs attributsΓintroduit au moins un inconvénient supplémentaire: celui de rendre délicate la modélisation d'attributs historiques *temporellement hétérogènes* au sein d'un même objet. Deux attributs historiques sont dits temporellement hétérogènes [CC93]Γ si les ensembles d'instants auxquels ils sont observés diffèrent. Cette différence peut être dû au fait que l'un des attributs est observé à un instant auquel l'autre ne l'est pasΓoù elle peut provenir d'une différence de granularité d'observation.

ConcrètementΓsupposons que l'attribut superviseur de la classe UnitéDeProduction est observé à la granularité de la minute (c.à.d. qu'une unité de production peut changer de superviseur plusieurs fois dans une journée)Γalors que les attributs production et qualité sont observés à la granularité du jour. La question se pose de savoir à quelle granularité doivent être données les estampilles des versions des objets de la classe UnitéDeProduction. Si ces estampilles sont données à la granularité du jourΓalors l'information sur les changements de superviseur au cours d'une journée sont perdues. Si par contre elles sont données à la granularité de la minuteΓalors les données relatives au volume et à la qualité de la production sont susceptibles d'être mal interprétées (le fait que le volume de production soit de 2000 au cours d'une journéeΓn'entraîne rien sur le volume de la production au cours d'une minute particulière).

Les versions d'objets constituent un mécanisme presque incontournable pour la prise en compte de l'évolution de schéma au niveau d'une base d'objets. En effetΓlorsque le schéma d'une base d'objets évolueΓet que les applications utilisant cette base nécessitent de pouvoir accéder aux objets créés sous l'ancien schémaΓil faut d'une façon ou d'une autre distinguer différentes versions d'un objetΓchacune possédant éventuellement son propre schéma [BF99]. AussiΓsi l'on veut historiser simultanément et orthogonalement l'évolution du schéma et celle des états des objetsΓtout en évitant les problèmes liés à la représentation des associations temporelles entraînés par les modèles à versionsΓil faut intégrer les concepts des modèles à versions avec ceux des modèles de données temporelles. À notre connaissanceΓcette intégration n'a été que très peu étudiée. Les quelques propositions sur ce sujet [RGMS99] n'ont pas été suivis par des implantations opérationnellesΓce qui réflète les difficultés techniques que leur mise en oeuvre au dessus d'un SGBD soulève.

# Récapitulatif

Malgré des débuts difficilesΓla technologie des "SGBD à objets" a atteint un certain degré de maturitéΓqui se reflète par la stabilisation des produits et leur convergence progressive vers une approche commune : celle prônée par le standard industriel ODMG.

Le standard ODMG comporte entre autresΓun langage de définition de schémasΓun

langage de requêtes déclaratif, et des passerelles pour les langages de programmation C++, Java et SmallTalk, le tout articulé autour d'un unique modèle de données fondé sur une philosophie "tout-objet". L'incompatibilité entre les systèmes de types des langages supportés par l'ODMG se reflète en des multiples incohérences dans ses versions successives. Le travail développé dans [Ala97] et [Ala99] détaille certaines de ces incohérences, et en propose des solutions. C'est dans le cadre du standard ODMG ainsi modifié que se place notre travail.

La modélisation logique et l'interrogation de données comportant des associations temporelles se révèlent des tâches relativement complexes lorsqu'on se restreint aux concepts et outils fournis par l'ODMG. En particulier, des requêtes temporelles très simples à formuler en langue naturelle, donnent lieu à des expressions très complexes dans le langage de requêtes OQL de l'ODMG. Cette complexité apparaît d'ailleurs dans beaucoup d'autres modèles et langages pour bases de données, qu'ils soient relationnels ou à objets.

Face à ce constat, de nombreux travaux de recherche se sont penchés sur l'intégration d'abstractions propres à la manipulation du temps, dans des modèles de données et des langages de requêtes existants (surtout relationnels). Ces travaux ont débouché sur une diversité d'extensions dites "temporelles" de ces modèles et langages. Bon nombre de ces propositions souffrent d'un défaut récurrent : celui de forcer l'utilisateur à manipuler une représentation fixée des associations temporelles, généralement à base d'intervalles. Nous avons montré que cette représentation n'est pas adaptée à l'expression d'une large classe de requêtes temporelles. De plus, dans le contexte des modèles à objets, cette pratique tend à violer le principe d'encapsulation.

L'un des principaux buts de notre thèse est d'unifier, enrichir et transposer les résultats de ces travaux, pour donner lieu à une extension temporelle du standard ODMG. Dans cet effort, une place prédominante est accordée à la prise en compte des principes et buts de la technologie à objets : modularité, encapsulation des représentations, prise en compte de l'aspect évolutif des applications, etc. En particulier, sur ce dernier point, l'accent est mis sur la migration transparente d'applications opérant sur une base de données classique, vers une base de données comportant des attributs et des associations temporels.

# Chapitre 3

# Le modèle d'objets temporels TEMPOS

Dans le chapitre précédent, nous avons mis en évidence certaines difficultés soulevées par la modélisation logique de données temporelles, et nous avons présenté quelques approches pour maîtriser la complexité qui en découle. Dans ce chapitre, nous nous proposons d'unifier ces approches, et d'en encapsuler les principales abstractions dans un système de types conforme à celui de l'ODMG. Notre démarche est progressive : nous commençons par des abstractions simples (instants, durées, ensembles d'instants), que nous composons ensuite avec celles fournies par le modèle ODMG, pour aboutir à des concepts plus élaborés : historiques, propriétés et classes temporelles. Enfin, nous étudions les problèmes posés par le passage d'une base de données conforme à l'ODMG, à une base de données intégrant les abstractions que nous proposons.

La section 3.1 reprend des idées développées dans [Can97], tout en les complétant par rapport à des besoins dégagés au cours de notre travail. Les sections 3.2 et 3.3 constituent notre contribution au modèle TEMPOS.

## 3.1 Types temporels et historiques

### 3.1.1 Modèle du temps

Le concept le plus élémentaire du modèle TEMPOS, est celui de *ligne de temps*. De façon abstraite, une ligne de temps est un couple $(\mathsf{D}, <_T)$ composé d'un ensemble fini de *chronons* $\mathsf{D}$, et d'une relation binaire $<_T$ définissant un ordre linéaire et total sur $\mathsf{D}$. Par la suite, on suppose sans perte de généralité, qu'une ligne de temps est un intervalle d'entiers dont la borne inférieure est zéro. De cette façon, chaque chronon est identifié par un entier naturel.

Du point de vue pratique, une ligne de temps peut servir à modéliser une portion discréti-

sée du temps "réel" comme par exemple l'ensemble de jours entre les 1er septembre 1900 et le 31 décembre 1999 ou alors l'ensemble de secondes entre le 1er janvier 1970 à 0h00:00 et le 31 décembre 2034 à 23:59:59. Comme nous le verrons dans le chapitre 5 on peut aussi utiliser une ligne de temps pour modéliser l'ensemble de *frames* dans une vidéo et plus généralement n'importe quelle suite d'événements linéairement ordonnés.

Suivant [CR87 WJS95] nous définissons une *granularité* sur une ligne de temps comme étant une partition de l'ensemble de chronons de cette ligne en des ensembles convexes appelés des *grains*. Par rapport au premier des exemples cités dans le paragraphe précédent le partitionnement en semaines mois et années correspondent à des granularités. Notons que dans cet exemple le jour correspond aussi à une granularité dans laquelle chacun des grains est un singleton. Plus généralement la granularité obtenue en partitionant une ligne de temps LDT en des singletons est appelée la *granularité minimale* de LDT.

Du fait qu'une granularité partitionne les chronons d'une ligne de temps en des ensembles convexes l'ordre défini sur les chronons induit un ordre linéaire et total sur les grains composant une granularité. Ceci permet en particulier de numéroter les grains d'une granularité par des entiers positifs (en partant de zéro par exemple) numérotation qu'on utilisera par la suite pour les identifier. Dans l'exemple ci-dessus le mois allant du 1er au 31 Janvier 1900 est alors identifié par l'entier zéro.

L'ensemble de granularités sur une même ligne de temps est structuré en une hiérarchie suivant une relation d'ordre partielle $\prec$ définie comme suit.

**Definition 1** (*Relation $\prec$, ou "plus fine que"*). Une granularité G1 est dite *plus fine* qu'une autre granularité G2 (G1 $\prec$ G2) ssi G1 et G2 sont définies sur la même ligne de temps et

$$\forall g_2 \in G2,\ g_2 = \bigcup_{g_1 \in G1,\ g_1 \subseteq g_2} g_1$$

Réciproquement on dit aussi que G2 est plus grossière que G1. □

En continuant avec notre exemple nous pouvons dire que la granularité mois est plus fine que l'année mais plus grossière que le jour. La granularité semaine n'est pas comparable avec la granularité mois ni avec l'année.

L'intérêt de la relation $\prec$ est de permettre les conversions entre des grains appartenant à différentes granularités. Plus précisément pour chaque couple de granularités G1 et G2 (G1 $\prec$ G2) nous pouvons définir deux fonctions de conversion : une permettant d'approximer un grain de G1 au travers du grain de G2 qui le contient et une autre permettant d'associer un ensemble (convexe) de grains de G1 à chaque grain de G2. Ces opérations sont respectivement appelées *approximation* ($\alpha$) et *expansion* ($\epsilon$) d'un grain. Elles sont illustrées dans la figure 3.1 et définies formellement comme suit :

- $\forall g_2 \in G2\ \ \epsilon_{\mathrm{G2,G1}}(g_2) = \{g_1 \in G1 \mid g_1 \subseteq g_2\}$

- $\forall g_1 \in G1 \;\; \alpha_{G1,G2}(g_1) =$ l'unique $g_2 \in G2$ tel que $g_1 \subseteq g_2$

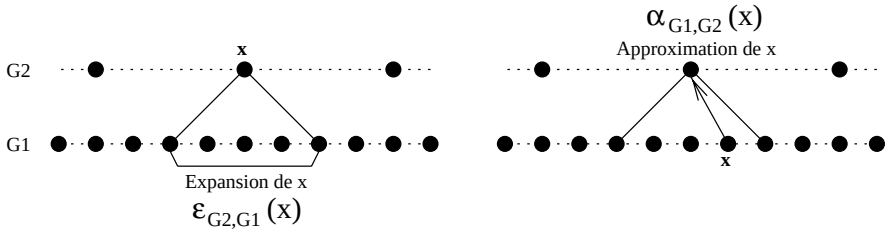



FIG. 3.1 – *Expansion et approximation d'un granule*

Selon les granularités concernées, les fonctions de conversion entre deux granularités sont plus ou moins complexes. Dans le cas du jour et de la semaine, il s'agit de fonctions linéaires, entièrement caractérisées par un facteur de conversion (7 ou 1/7 selon le sens de la conversion). On dit alors que la granularité semaine est *régulière* par rapport au jour. Dans le cas du jour et de l'année, la fonction de conversion fait intervenir des divisions euclidiennes par 4, 100 et 400, afin de tenir compte des années bissextiles (Cf. par exemple [GLOS97]).

Sur la base du concept de granularité ci-dessus, nous définissons un ensemble de types abstraits de données modélisant les valeurs temporelles manipulées par la suite.

- **Instant.** Un *instant* est une approximation d'un segment d'une droite de temps, par le biais d'un grain. Il est entièrement décrit par la donnée d'une granularité et d'un entier positif, appelé sa *position*.
  Au niveau conceptuel, la distinction entre *grain* et *instant* est très subtile : un grain n'est rien d'autre qu'un ensemble de chronons, alors qu'un instant est une entité en elle-même, associée à un grain mais indépendante des chronons qui composent ce grain. En particulier, dire qu'un fait a lieu à un instant, ne veut pas forcément dire que ce fait a lieu à chaque chronon du grain associé à cet instant (Cf. § 3.1.2).
- **Durée.** Une *durée* est un nombre de grains utilisé comme mesure de la distance entre deux instants. Pour permettre l'expression de déplacements dans le temps par rapport à un instant donné, que ce soit vers le passé ou vers le futur, nous considérons des *durées signées*, caractérisées par un entier relatif (leur *mesure*) et par une granularité.
- **Ensemble d'instants ou EDI.** Un EDI est composé d'instants de même granularité. Le type Intervalle est un sous-type d'EDI modélisant des ensembles d'instants convexes. L'introduction de ce sous-type est motivée par ses propriétés : fermeture par rapport à l'intersection, ensemble de relations de comparaison se réduisant à 13 relations élémentaires [All83], possibilité de caractériser un intervalle à partir de deux instants, etc.

D'autres types que nous ne décrivons pas ici, incluent ceux permettant de décrire les instants par leurs "coordonnées" dans un *système de granularités*, ou par rapport à un *for-*

*mat* [CD97]. Grâce à ces types, il est possible de dénoter le $36495^{\text{ème}}$ jour depuis le $1^{\text{er}}$ Janvier 1900 au travers de la liste d'entiers [1999, 12, 3] (par rapport au système de granularités [Année, Mois, Jour]), ou tout simplement au travers de la chaîne de caractères "3/12/1999" comme c'est l'usage. Le concept de format permet aussi de prendre en compte la notion de zone horaire (dans le cas des instants définis à des granularités au moins aussi fines que l'heure).

Chacun de ces types est muni d'un jeu d'opérateurs indépendant des représentations utilisées. Ce dernier point constitue d'ailleurs l'une des originalités de notre proposition. En effet, dans la majorité des travaux existants, les intervalles sont directement manipulés au travers de leur représentation sous forme de deux instants, et les ensembles d'instants sont identifiés à des ensembles d'intervalles disjoints "maximaux" (appelés *temporal elements* [JD98]). Nous soutenons que, même si ces représentations se révèlent souvent adéquates, le fait de les imposer dès le niveau logique est parfaitement inutile et induit une séparation entre le concept modélisé et les abstractions fournis par le modèle de données. Aussi, nous laissons toute discussion sur les aspects représentation de valeurs temporelles pour le chapitre 6, portant sur l'implantation.

Parmi les opérateurs sur les types temporels (dont on pourra trouver des descriptions in extenso dans [CD97, Can97]) on peut citer :

- Sélecteurs de l'instant minimum, maximum et de la durée d'un EDI. Par exemple :
  Minimum({ '13/5/98', '16/5/98', '18/5/98'}) = '13/5/98' et
  Durée(['13/5/98'..'16/5/98']) = '4 jours'.
- Relations d'ordre sur les instants et les durées ($<, >, =$).
- Opérateurs arithmétiques entre les instants et les durées : ajout et soustraction d'une durée à un instant, ajout et soustraction de deux durées. Par exemple :
  '1/2/90' + ' 3 jours' = '4/2/90'.
- Opérateurs de conversion entre des valeurs temporelles observées à différents niveaux de granularité : approximation d'un instant par un autre, expansion d'un instant en un intervalle, et conversion d'une durée de granularité G1 en une durée de granularité G2 (G1 régulière par rapport à G2). Par exemple :
  Conversion('2 jours', Heure) = '48 heures'.
  Les opérateurs d'approximation et d'expansion sur des instants sont des extensions directes de ceux définis sur les grains (Cf. figure 3.1). Par exemple :
  Approximation('1/2/90', Mois) = 'Février 90'
  Expansion('1/2/90', Minute) = ['1/2/90 à 0h00'..'1/2/90 à 23h59:59'].
- Opérateurs ensemblistes sur les EDI (union, intersection, différence).
- Opérateurs de comparaison entre deux intervalles, dans la lignée de ceux définis dans [All83] : précède, chevauche, contient, etc.

Nous introduirons d'autres opérateurs sur les types temporels dans le chapitre suivant lors de la présentation du langage de requêtes TEMPOQL. Cela nous permettra de décrire simultanément leur syntaxe dans ce langage et leur sémantique.

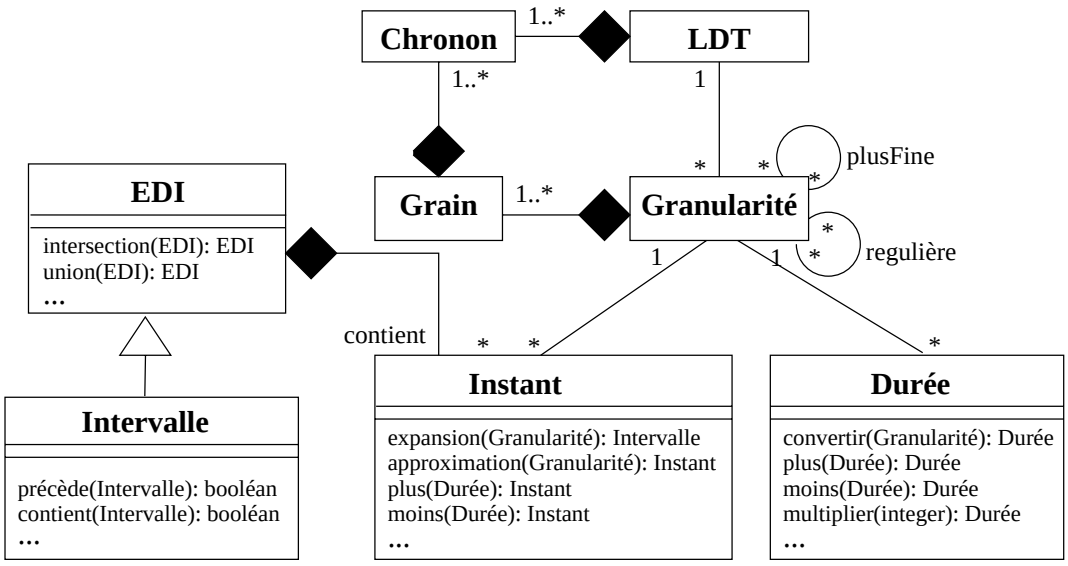



FIG. 3.2 – *Modélisation UML des types temporels*

La figure 3.2 fournit un modèle UML des types temporels. Ces types temporels complètent ceux déjà fournis par le modèle ODMG (à savoir Date, Timestamp, Interval et Time), dont la sémantique est définie par rapport à des lignes de temps et des granularités fixées par le standard.

### 3.1.2 Modèle d'historiques

Le modèle d'historiques de TEMPOS est fondé sur un seul type paramétré Historique<T>. De façon abstraite, une valeur de type Historique<T> dénote une fonction à domaine dans un ensemble fini d'instants de même granularité, et à image dans un ensemble d'objets de type T.

Tout comme les types temporels introduits précédemment, le type Historique est muni d'un jeu d'opérateurs comportant des constructeurs, des sélecteurs et des opérateurs algébriques. Ces opérateurs sont spécifiés dans l'annexe A et présentés informellement dans la suite.

Parmi les diverses représentations des historiques, nous en distinguons deux :

- La *représentation par instants* ou par *IChronique*. Représentation sous forme d'une liste de couples ⟨ instant, objet ⟩ triée en ordre chronologique. Par exemple :
  [⟨1, o1⟩, ⟨2, o1⟩, ⟨4, o1⟩, ⟨5, o2⟩, ⟨6, o2⟩, ⟨7, o2⟩, ⟨8, o3⟩, ⟨9, o1⟩, ⟨10, o1⟩].
- La *représentation par intervalles* ou par *XChronique*. Représentation sous forme d'une liste de couples ⟨ intervalle, objet ⟩ avec des intervalles disjoints, maximaux et chrono-

logiquement ordonnés. Par exemple :
[⟨[1..2], o1⟩, ⟨[4..4], o1⟩, ⟨[5..7], o2⟩, ⟨[8..8], o3⟩, ⟨[9..10], o1⟩].

Dans certaines situations, ces deux modes de représentation se révèlent adaptés à la manipulation d'historiques dans des programmes et à la formulation de requêtes. À cet effet, des constructeurs et des sélecteurs permettant d'obtenir un historique à partir d'une de ces représentations et vice-versa sont fournis. Concrètement, le sélecteur IChronique permet d'obtenir la représentation par instants d'un historique. Inversement, l'opérateur IHistorique construit un historique à partir d'une collection de couples ⟨ instant, objet ⟩ vérifiant certaines propriétés (Cf. annexe A). De façon analogue, le sélecteur et le constructeur associés à la représentation par intervalles sont appelés XChronique et XHistorique respectivement.

Parmi les autres sélecteurs du type Historique, nous pouvons citer :

- Les opérateurs Domaine(H) et Image(H) permettant d'accéder respectivement au domaine et à l'image de l'historique H vu comme une fonction. Le résultat du sélecteur Domaine est de type EDI, et celui d'Image est un ensemble d'objets.
- L'opérateur Valeur(H, I) permettant d'obtenir la valeur de l'historique H à l'instant I, à condition que I appartienne au domaine de H.

### 3.1.3 Opérateurs algébriques sur les historiques

À des fins de présentation, nous classons les opérateurs algébriques sur les historiques en intra-points et inter-points. Un opérateur est dit *intra-point* si la valeur de son résultat à un instant donné dépend exclusivement de la (des) valeur(s) du (des) argument(s) à ce même instant. Dans le cas contraire, il est dit *inter-point*.

#### Opérateurs intra-points

Les opérateurs intra-points du modèle d'historiques TEMPOS (Cf. [FCS97, Can97, SFC98]) s'inspirent des opérateurs de sélection, projection et produit naturel de l'algèbre relationnelle.

Un historique étant vu comme une fonction, et a fortiori comme une relation binaire, l'opérateur de sélection de l'algèbre relationnelle permet de restreindre cet historique aux instants où il vérifie une condition donnée. Dans notre modèle d'historiques, nous particularisons le cas où cette condition porte uniquement sur l'attribut dénotant le domaine de l'historique (*restriction selon le domaine*), de celui où elle porte uniquement sur l'attribut correspondant à l'image (*restriction selon l'image*). En outre, dans le cas d'une restriction selon le domaine, nous nous restreignons aux conditions d'appartenance d'un instant à un EDI. Les opérateurs ainsi obtenus sont respectivement notés $\Sigma_{\text{dom}}$ et $\Sigma_{\text{im}}$, et illustrés sur un exemple dans la figure 3.3.

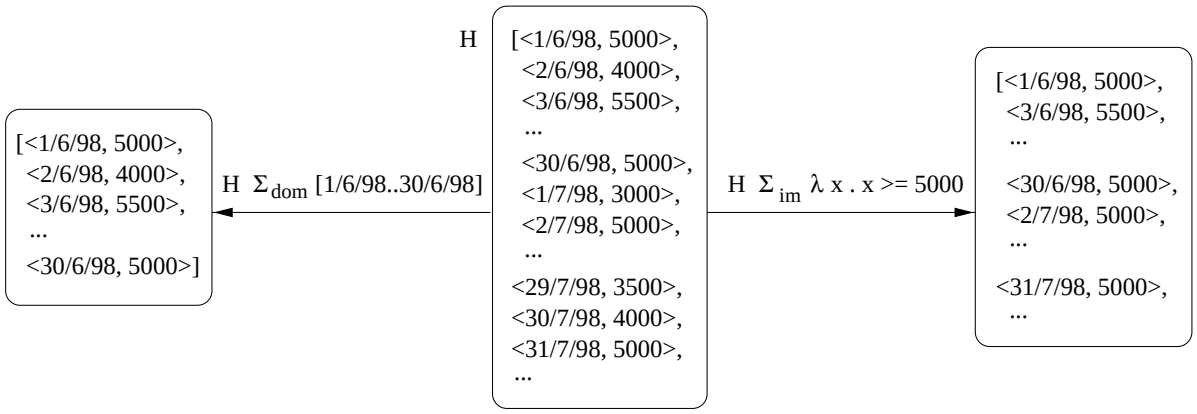



FIG. 3.3 – *Restrictions selon le domaine ( $\Sigma_{dom}$ ) et selon l'image ( $\Sigma_{im}$ )*

Dans la même lignée, le produit naturel relationnel peut aussi être étendu aux historiques en prenant comme attributs de jointure ceux dénotant les domaines des historiques arguments (Cf. figure 3.4). Ceci conduit à la définition d'un opérateur H1 $*_{\cap}$ H2 construisant un historique par juxtaposition des valeurs "synchrones" de H1 et de H2. En prenant H1 et H2 de type Historique<T1> et Historique<T2> respectivement, l'image de H1 $*_{\cap}$ H2 est alors formée de couples $\langle$ o1, o2 $\rangle$ tels que o1 est de type T1 et o2 de type T2. Par construction, le domaine de H1 $*_{\cap}$ H2 est l'intersection des domaines de H1 et de H2.

Alternativement, on peut appliquer la démarche ci-dessus au produit naturel externe symétrique [DA83], et déboucher sur un opérateur similaire au précédent, hormis le fait que le domaine de l'historique résultat est l'union des domaines des historiques arguments. L'opérateur ainsi défini est appelé *produit d'historiques externe* et noté H1 $*_{\cup}$ H2. Par convention, aux instants où l'un des deux historiques n'est pas défini alors que l'autre l'est, la valeur de H1 $*_{\cup}$ H2 est de la forme $\langle$ o1, Nil $\rangle$ (resp. $\langle$ Nil, o2 $\rangle$), où Nil est la valeur "neutre" du type des valeurs de H2 (resp. H1).

Toujours dans le but d'étendre les opérateurs de l'algèbre relationnelle aux historiques, nous rappelons que l'opérateur de projection relationnelle se transpose dans les modèles de données à objets en un opérateur d'application d'une fonction (a priori quelconque) à chaque élément d'une collection. Une extension naturelle de ce dernier opérateur aux historiques consiste à dériver un historique H' par application d'une fonction à chaque valeur prise par H. Par construction, le domaine de H' est alors le même que celui de H. Suivant une dénomination très répandue dans le cadre des langages de programmation fonctionnels [CM95], nous appelons l'opérateur résultant Map. Cet opérateur est illustré dans la figure 3.5.

D'autres opérateurs intra-points s'obtiennent en appliquant les opérateurs d'intersection ou de différence ensemblistes aux historiques dénotés par leurs graphes. Ceci est possible car l'intersection (ou la différence) des graphes de deux historiques est lui-même le graphe

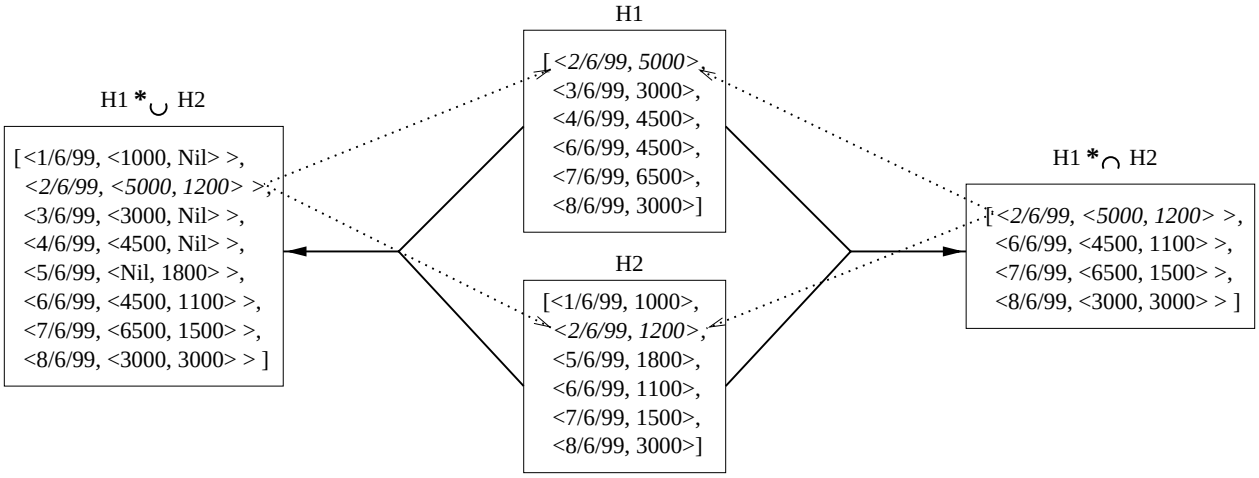



FIG. 3.4 – *Produit d'historiques interne et externe*

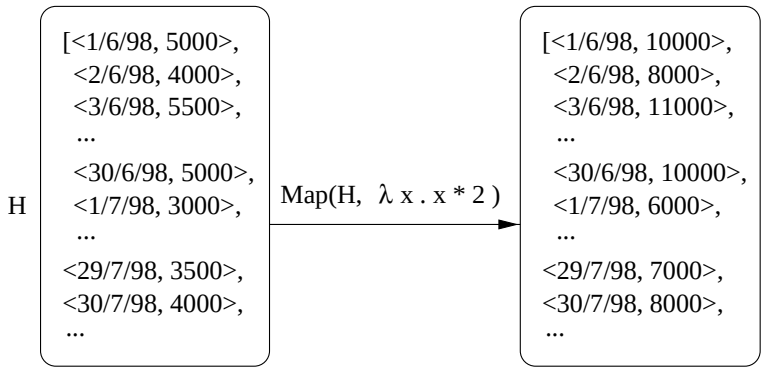



FIG. 3.5 – *Application d'une fonction point-par-point (opérateur* Map*).*

d'un historique. Malheureusement, cette propriété ne s'applique pas à l'union. En effet, soit H1 et H2 deux historiques dont les valeurs à un instant I sont respectivement V1 et V2. Quelle serait alors la valeur de H1 ∪ H2 à l'instant I ? En choisissant par convention V2, on obtient un opérateur d'union asymétrique noté $\cup_+$, que nous retenons dans le modèle car il intervient dans la définition de certains opérateurs de mises à jour sur les propriétés temporelles (Cf. § 3.2.1).

### Opérateurs inter-points

Les opérateurs inter-points du modèle TEMPOS (Cf. [DFS98, DFS99a]), comprennent ceux liés au regroupement et à l'agrégation, ainsi que des opérateurs permettant de raisonner sur la succession dans le temps en découpant un historique en deux selon un prédicat.

S'agissant des opérateurs de regroupement, le modèle TEMPOS en comprend deux. Le premier, noté , permet d'exprimer des changements de granularité, en structurant un his-

torique suivant le partitionnement de la droite de temps induit par une granularité donnée. Étant donnés un historique H et une granularité G plus grossière que la granularité de H H Γ G est un historique de granularité GΓdont les valeurs sont des sous-historiques disjoints de H (voir figure 3.6).

Le deuxième opérateur de regroupement d'historiquesΓnoté H Δ ⟨ D1, D2 ⟩Γpermet d'obtenir à partir d'un historique H et de deux durées positives D1 et D2Γdes sous-historiques de H dont les domaines sont des intervalles de taille D1 + D2 + 1. Plus précisémentΓH Δ ⟨ D1, D2 ⟩ est l'historique dont la valeur à l'instant I est le sous-historique obtenu par restriction de H à l'intervalle [I − D1..I + D2]Γpourvu que [I − D1..I + D2] ⊆ Domaine(H) (Cf. figure 3.6).

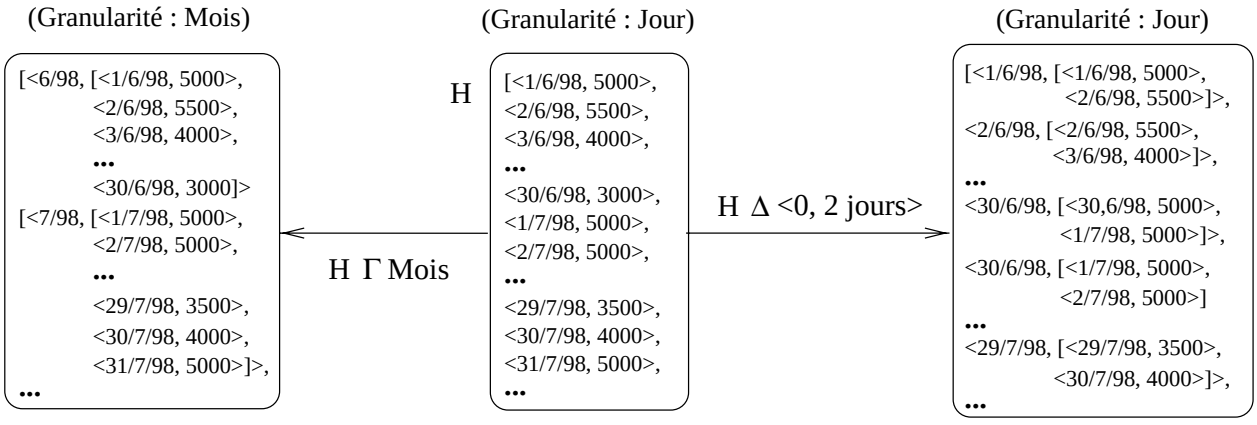



FIG. 3.6 – *Opérateurs de regroupement*

Tout comme l'opérateur Γ le résultat de Δ est un historique d'historiques. CependantΓà la différence de ΓΓl'historique construit par Δ est défini à la même granularité que l'historique de départ. De plusΓles sous-historiques obtenus à partir de Δ ne sont pas forcément disjoints.

S'agissant d'opérateurs d'agrégation numériqueΓle modèle TEMPOS fournit des versions "cumulatives" des opérateurs minΓmaxΓsum et avg présents dans SQL et OQL. Le principe de ces opérateurs est de construire un historique qui à chaque instant I dans le domaine de HΓassocie l'agrégation des valeurs prises par H aux instants précédant I. Par exempleΓen supposant que H est l'historique de la production journalière d'une unité entre le 1/1/99 et le 31/5/99Γl'agrégation cumulative SommeC(H)Γfournit pour chaque jour sur cet intervalleΓ la somme des productions de cette unité depuis le début de l'année. Notons qu'à partir de ce dernier opérateur et des sélecteurs sur les historiquesΓon peut décrire un opérateur SommeΓ qui calcule la somme de toutes les valeurs (pas forcément distinctes)Γprises par un historique:

Somme(H) = Valeur(SommeC(H), Max(Domaine(H))).

La spécification des opérateurs d'agrégation cumulative ci-dessusΓne va pas sans poser des problèmes de typage. En effetΓces opérateurs n'ont de sens que pour des historiques à valeurs

numériquesΓalors qu'ils sont définis au niveau du générateur de types Historique qui peut être instancié avec n'importe quel type. Pour tenir compte de cette particularitéΓnous proposons d'utiliser le concept de *type paramétré contraint* (Cf. § 2.1.2). Plus précisémentΓpour chaque type numérique du modèle ODMG (floatΓdoubleΓshortΓlongΓetc.)Γnous introduisons un sous-type de Historique<T>Γsur lequel nous contraignons le paramètre à être un sous-type du type numérique en question. Dans le cas du type float par exempleΓun type HistoriqueFlottant<T> est définiΓ et associé à la contrainte "T sous-type de float". Les opérateurs d'agrégation cumulative sont alors définis sur chacun de ces générateurs de types contraints.

La dernière famille d'opérateurs que nous considérons permettent de raisonner sur la succession dans le tempsΓpar exemple en restreignant un historique aux instants suivant le plus petit instant où il satisfait une condition donnée. En remplaçant le mot "suivant" par "précédent" et l'expression "plus petit" par "plus grand" dans cette définitionΓon obtient quatre opérateurs que nous appelons AprèsPremierΓAvantPremierΓAprèsDernier et AvantDernier[1]. Ces opérateurs sont tous paramétrés par un prédicat (voir figure 3.7).

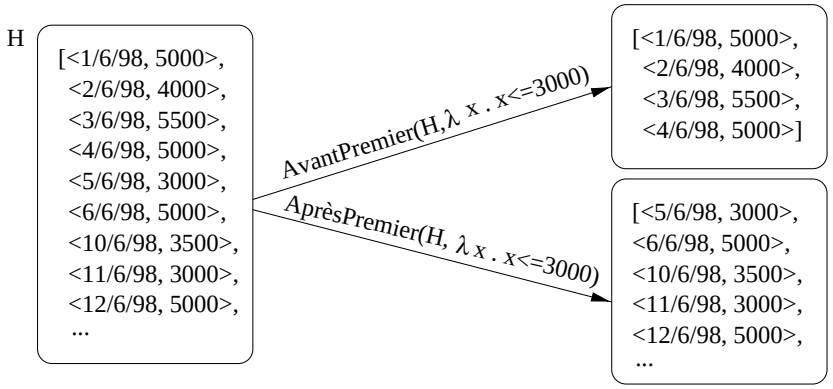



FIG. 3.7 – *Opérateurs de découpage d'historiques selon un prédicat portant sur les valeurs d'un historique.*

Pour résumerΓla figure 3.8 fournit un aperçu des opérateurs algébriques sur les historiques.

## 3.2 Classes et propriétés temporelles

En utilisant les types du modèle du temps et du modèle d'historiques décrits ci-dessusΓil est possible de modéliser l'évolution d'un phénomèneΓpar exempleΓau travers d'une propriété de type Historique<T> (pour un certain type T). AinsiΓla base de données exemple étudiée dans la section 2.2Γpeut être modélisée à l'aide des historiques (Cf. figure 3.9).

Cette approche possède un certain nombre de limitations que nous décrivons par la suite.

1. Le lecteur spécialiste pourra remarquer que ces opérateurs correspondent à des versions algébriques des modalités $\Diamond$ et $\blacklozenge$ des logiques temporelles linéaires [Eme90].

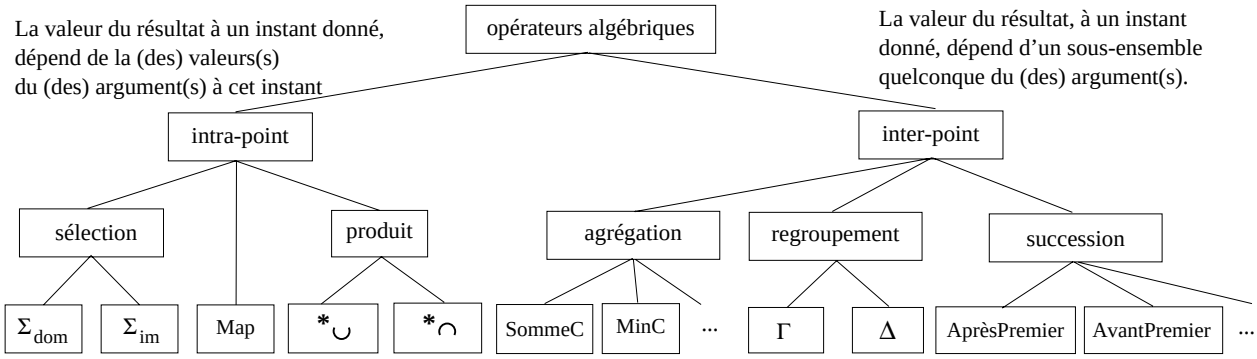



FIG. 3.8 – *Taxinomie des opérateurs algébriques sur les historiques*

```
class UnitéDeProduction(extent LesUnités, key codeUnité) {
   attribute string codeUnité;
   attribute Historique<Superviseur> hist_superviseur;
   attribute Historique<set<Ouvrier>> hist_ouvriers;
   attribute Historique<short> hist_production;
   attribute Historique<float> hist_qualité;
}
class Employé (extent LesEmployés, key (nom, dateNaissance)) {
   attribute string nom;
   attribute Instant dateNaissance;          /* Alternativement on pourrait utiliser le type Date */
   attribute Historique<float> hist_salaire;
}
class Ouvrier extends Employé (extent LesOuvriers) {
   attribute Historique<UnitéDeProduction> hist_unité;
}
class Superviseur extends Employé (extent LesSuperviseurs) {
   attribute Historique<UniteDeProduction> hist_supervisions;
}
```

FIG. 3.9 – *Schéma de la base de données exemple en ODL étendu par le type historique.*

Primo, tout comme dans le schéma ODL de la figure 2.2(b) page 23, les contraintes d'intégrité référentielles sur les paires de propriétés modélisant des associations temporelles (p. ex. hist_superviseur et hist_supervisions) ne sont pas spécifiées dans le schéma. Ce schéma n'intègre donc pas la contrainte *"à tout instant, le superviseur d'une unité supervise cette unité"*.

Secundo, dans cette approche la prise en compte de l'impact d'un changement dans un phénomène observé, sur l'historique de la propriété qui le modélise, est entièrement à la charge des programmes d'application. Cette situation est problématique, en particulier lorsque l'évolution du phénomène observé a lieu en parallèle avec l'évolution de la base de données, comme c'est le cas notamment lorsqu'on observe une propriété selon le temps de transaction (Cf. § 2.3.1). Concrètement, supposons que l'on veuille observer l'association entre les unités

de production et leurs superviseurs selon le temps de transaction. Autrement dit, on vise à observer l'évolution de la connaissance que possède la base de données sur cette association (quand l'affectation de tel superviseur à telle unité a-t-elle été enregistrée?), et non pas l'évolution de l'association dans le monde modélisé (quand l'affectation de tel superviseur à telle unité a-t-elle eu lieu?).

Si l'on utilise des historiques pour modéliser cette association, comment traduire dans l'historique de la propriété hist_superviseur, le fait que le superviseur d'une unité le reste jusqu'à nouvel ordre? En d'autres termes, comment représenter le fait que tel employé est superviseur de telle unité de production depuis une certaine date, et jusqu'à une date indéterminée dans le futur[2]? Comment assurer ensuite que les connaissances contenues dans les historiques des propriétés hist_superviseur et hist_supervision évoluent de façon monotone[3]? En effet, l'une des caractéristiques des propriétés observées selon le temps de transaction est justement que l'on ne remet jamais en cause les connaissances que l'on a sur le passé (Cf. § 2.3.1).

Tertio, dans l'exemple ci-dessus, il n'est pas possible de fixer au niveau du schéma la granularité à laquelle chacune des propriétés temporelles est observée. En effet, il est parfaitement possible que pour une unité de production donnée, sa production soit observée à la granularité de la minute, alors que pour une autre, elle le soit à la granularité du mois. Cela entraîne des problèmes lorsqu'on veut comparer les productions de ces deux unités.

Notre réponse à ces limitations consiste à étendre les concepts de "classe" et de "propriété" tels que définis dans le modèle ODMG, pour donner naissance à des concepts de classe et de propriété temporelles, répondant aux besoins mis en évidence ci-haut. Cette approche est similaire à celles qui ont été adoptées dans TOOBIS et TAU (Cf. § 2.3.3). Elle se distingue par contre de celle adoptée dans T_ODMG, où les attributs temporels sont modélisés comme dans l'exemple ci-dessus, à ceci près que la granularité des historiques peut être spécifiée dans le schéma (Cf. [MBFG99]).

### 3.2.1 Historisation des valeurs des propriétés

Nous rappelons que dans le modèle d'objets de l'ODMG, une propriété est un concept modélisant une caractéristique commune à tous les objets d'une classe. De même que les classes, les propriétés peuvent être instanciées, même si le mécanisme d'instantiation des propriétés diffère de celui des objets. En effet, les objets sont instanciés explicitement au travers de l'opérateur new, alors que les propriétés sont instanciées implicitement lors de la création d'un objet. Par ailleurs, les instances de propriétés ne sont pas des entités à

2. On rappelle qu'en TSQL2, ceci est modélisé au travers d'une constante UC (Cf. page 31).
3. C'est-à-dire qu'on n'efface jamais de l'information.

part entière, au même titre que les objets. En particulier, une instance de propriété n'a pas d'identificateur, et il n'est pas possible d'avoir des "propriétés de propriétés".

En TEMPOS, on distingue les propriétés dites *temporelles* des propriétés *fugitives*. Une propriété est temporelle si ses valeurs successives sont significatives et donc stockées. Elle est fugitive dans le cas contraire. Lorsqu'une propriété est temporelle, chacune de ses instances possède un historique, dont la granularité est fixée par une caractéristique de la propriété appelée sa *granularité d'observation*. Quelle que soit sa granularité, l'historique d'une propriété temporelle modélise son évolution soit par rapport au monde modélisé, soit par rapport au contenu de la base. La *dimension temporelle* d'une propriété temporelle ("temps de validité" ou "temps de transaction"), détermine laquelle de ces deux optiques est adoptée. Dans le cas dégénéré où le temps de transaction coïncide avec le temps de validité (Cf. § 2.3.1), la dimension temporelle "temps de transaction" peut être utilisée.

Tout comme les propriétés fugitives, une propriété temporelle possède un *type*. Dans le cas des propriétés fugitives, ce type modélise le domaine des valeurs que peut prendre une instance de cette propriété. Dans le cas d'une propriété temporelle, il modélise le domaine des valeurs que peut prendre l'historique attaché à une instance de cette propriété. Ainsi, si le type d'une propriété temporelle est Employé, l'historique d'une instance de cette propriété est de type Historique<Employé>.

À chaque instance d'une propriété temporelle est associé un ensemble d'instants appelé son *domaine d'observation*. Le domaine d'observation d'une instance de propriété temporelle détermine le domaine de son historique. La valeur de cet historique à un instant appartenant à ce domaine est soit définie directement par les mises à jour ayant eu lieu sur cette instance de propriété, soit calculée à partir des données fournis par ces mises à jour.

Plus précisément, les mises à jour sur une instance de propriété temporelle construisent un historique que nous qualifions d'*effectif*. Le domaine de cet historique est alors appelé le *domaine effectif* de l'instance de propriété considérée, et la différence entre le domaine d'observation de cette instance, et son domaine effectif, est appelé son *domaine potentiel*.

La valeur de l'historique d'une instance de propriété temporelle à un instant appartenant à son domaine potentiel, est calculée au travers d'une *modalité d'interpolation* attachée à la propriété temporelle concernée. Dans le cas des propriétés de validité, TEMPOS reconnaît trois modes d'interpolation (voir figure 3.10) :

- Discret : les valeurs de l'historique aux instants du domaine potentiel sont toutes égales à la valeur neutre du type de cette propriété, ou à une autre *valeur par défaut* fournie par l'utilisateur. Cette modalité est utilisée pour modéliser des phénomènes tels que le volume de production d'une unité : l'ensemble des instants pendant lesquels cet attribut est défini est connu à l'avance (tous les jours ouvrables), mais il peut arriver que pour certains jours ouvrables aucune valeur ne soit saisie, dans quel cas il est naturel de dire

(par exemple) que sa valeur est 0.

- En escalier : la propriété considérée évolue de telle manière que ses valeurs restent stables entre deux instants dans le domaine effectif. Ceci est le cas par exemple du salaire d'un employé. Tout comme pour les propriétés discrètes, la donnée d'une valeur par défaut est nécessaire dans certaines situations, par exemple lorsque le premier instant dans le domaine effectif est strictement supérieur au premier instant dans le domaine d'observation, comme c'est le cas dans l'exemple de la figure 3.10.
- Linéaire : entre deux valeurs successives dans le domaine effectif, la valeur de l'historique varie linéairement (c.à.d. avec une pente constante). Cette modalité d'interpolation ne s'applique qu'à des valeurs temporelles de type numérique. Cette restriction est prise en compte dans le méta-modèle de TEMPOS au travers du concept de classe paramétrée contrainte (Cf. § 2.1.2).

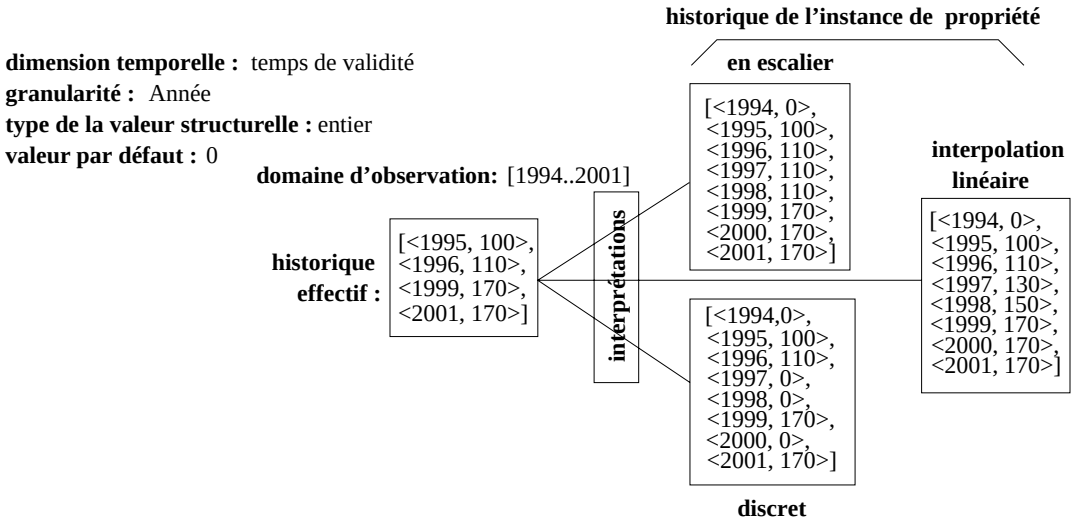



FIG. 3.10 – *Calcul de l'historique d'une propriété de validité.*

La modalité d'interpolation des propriétés de transaction est toujours celle en escalier (Cf. figure 3.11). La borne supérieure du domaine d'observation de ces propriétés est l'instant associé à l'horloge du système. Nous reviendrons sur ce point dans le paragraphe sur la mise à jour des propriétés de transaction.

Dans le modèle d'objets de l'ODMG, chaque instance d'une propriété possède une valeur qu'il est possible d'accéder et de modifier, modulo les restrictions imposées par le mécanisme d'encapsulation. Une façon de modéliser cette situation, consiste à dire qu'il existe au niveau du schéma de toute base de données ODMG, une interface que nous appelons InstanceDePropriété, sur laquelle sont définies deux méthodes : obtenirValeur et modifierValeur. La première de ces méthodes permet d'accéder à la valeur d'une instance de propriété, alors que

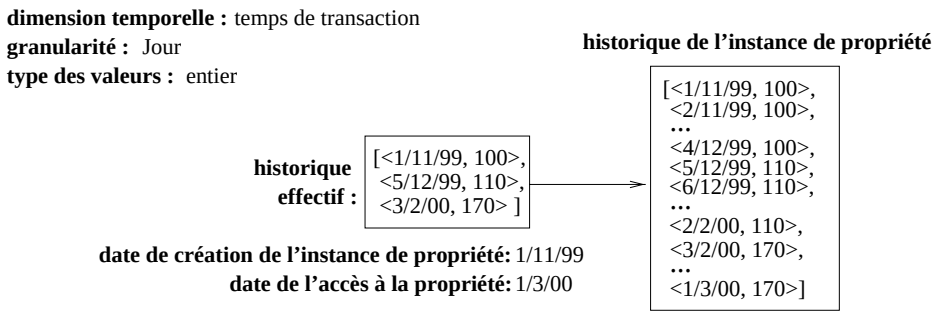



FIG. 3.11 – *Calcul de l'historique d'une propriété de transaction.*

la deuxième permet de la modifier. Cette vision *comportementale*[4] du modèle d'objets de l'ODMG est d'ailleurs présente dans les premières spécifications du standard. En particulier dans [Cat94] l'interface InstanceDePropriété apparaît sous le nom de Property et les méthodes obtenirValeur et modifierValeur sont respectivement nommées get_value et set_value. Bien que ces interfaces soient absentes des versions plus récentes du standard [CB97] nous les retenons dans la variante de l'ODMG que nous considérons car cela permet de spécifier plus finement la nature et le comportement des propriétés.

Concrètement dans cette modélisation comportementale des propriétés une propriété particulière (p. ex. la propriété unité de la classe Ouvrier) correspond à une classe implantant l'interface InstanceDePropriété. Les instances de cette classe modélisent des liens entre des objets des classes Ouvrier et UnitéDeProduction. Bien entendu l'interface InstanceDePropriété et toutes les classes qui l'implantent sont cachées aux programmes d'application et aux utilisateurs qui ne perçoivent les propriétés qu'au travers des notations O.P et O.P = ... fournies par les langages de requêtes et de programmation au travers desquels ils accèdent à la base. En ce sens les instances de propriétés ne sont pas traitées comme des entités à part entière qu'il serait possible de manipuler indépendamment de l'objet auquel elles sont attachées. Ceci rend la modélisation comportementale des propriétés compatible avec la modélisation qui en est faite dans le modèle d'objets de l'ODMG.

Toujours suivant la même optique nous modélisons les propriétés temporelles sous forme de classes implantant une interface que nous appelons InstanceDePropriétéTemporelle qui hérite de l'interface InstanceDePropriété. Les propriétés fugitives quant à elles correspondent à des classes implantant l'interface InstanceDePropriété mais n'implantant pas l'interface InstanceDePropriétéTemporelle.

Pour tenir compte de la dichotomie temps de validité vs. temps de transaction nous spécialisons l'interface InstanceDePropriétéTemporelle en deux sous-classes InstanceDePropriété-DeTransaction et InstanceDePropriétéDeValidité. L'interface InstanceDePropriétéDeValidité est

4. L'adjectif "comportemental" est utilisé pour marquer le fait que les concepts de base utilisés sont ceux d'entité et de méthode : les propriétés étant vues comme des entités possédant des méthodes particulières.

à son tour spécialisée en InstanceDePropriétéDiscrete, InstanceDePropriétéEscalier et InstanceDePropriétéLinéaire, correspondant aux trois modalités d'interpolation introduites ci-dessus. L'interface InstanceDePropriétéDeTransaction n'a en revanche pas de sous-interfaces. Dans la suite, nous détaillons les méthodes définies sur chacune de ces interfaces.

### Mise à jour des propriétés de transaction

Étant donné qu'une propriété temporelle observée suivant le temps de transaction modélise l'évolution d'une partie d'une base de données, le domaine d'observation d'une instance de propriété de ce type évolue automatiquement avec l'horloge du système. Conceptuellement, ceci revient à dire qu'à chaque top de l'horloge du système, sa nouvelle valeur, approximée à la granularité de la propriété de transaction en question, est rajoutée au domaine d'observation de chaque instance de cette propriété.

Il est possible de suspendre l'évolution automatique du domaine d'observation d'une instance de propriété de transaction. Ceci permet de modéliser le fait que la propriété en question n'est pas pertinente pendant une certaine période. Dans l'exemple qui nous occupe, une telle situation peut avoir lieu si une unité de production est supprimée temporairement : l'attribut temporel superviseur (observé selon le temps de transaction) n'est pas pertinent pendant la période d'absence de cette unité.

Le mécanisme de contrôle de l'évolution automatique des instances d'une propriété de transaction est fondé sur deux méthodes activer et suspendre, avec les sémantiques intuitives.

L'historique effectif d'une propriété de transaction évolue au travers d'appels successifs à la méthode modifierValeur dont la signature est héritée de l'interface InstanceDePropriété. La sémantique de cette méthode est bien sûr "surchargée" au niveau de l'interface InstanceDePropriétéDeTransaction. Plus précisément, étant donnée une instance de propriété de transaction IPT, IPT.modifierValeur(o) remplace l'historique effectif de IPT par un autre, identique au précédent à l'exception qu'il associe l'objet o à l'instant associé à la transaction où cette méthode est invoquée.

### Mise à jour des propriétés de validité

À la différence des propriétés de transaction, le domaine d'observation des propriétés de validité ne varie pas automatiquement avec l'horloge du système. Sa valeur est au contraire fixée par les programmes d'application au travers d'un opérateur noté modifierDomObservation, qui prend en argument un ensemble d'instants. Puisque le domaine d'observation doit contenir le domaine effectif, cet opérateur peut, dans certains cas, modifier l'historique effectif associé à la propriété. Par exemple, si cette contrainte est violée suite à une mise à jour du domaine d'observation, l'historique effectif est restreint au nouveau domaine d'observation.

L'historique effectif quant à lui est fixé au travers de l'opérateur modifierHistoEffectif prenant en argument un historique. Par ailleurs dans le but de faciliter la migration d'applications de l'ODMG vers TEMPOS la sémantique de l'opérateur modifierValeur lorsqu'il opère sur une propriété de validité (soit IPV) est définie comme suit :

IPV.modifierValeur(o) ≡ IPV.modifierHistoEffectif(IPV.obtenirHistoEffectif()
$\cup_+$ IHistorique(bag(struct(VT: instant_present()[5], VS: o))))

Où IPV.obtenirHistoEffectif() dénote l'historique effectif de l'instance de propriété IPV et $\cup_+$ est l'opérateur d'union asymétrique d'historiques introduit dans le § 3.1.3.

Par exemple soit IPV l'instance de propriété de validité de la figure 3.10. Si la mise à jour IPV.modifierValeur(100) est effectuée au cours de l'année 2000 son historique effectif devient :

{ ⟨ 1995, 100 ⟩, ⟨ 1996, 110 ⟩, ⟨ 1999, 170 ⟩, ⟨ 2000, 100 ⟩, ⟨ 2001, 170 ⟩ }

De façon analogue à modifierDomObservation l'opérateur modifierHistoEffectif peut modifier le domaine d'observation de la propriété de validité sur laquelle il s'applique afin de renforcer la contrainte d'inclusion du domaine effectif dans le domaine d'observation. Plus précisément lorsque suite à une mise à jour de l'historique effectif cette contrainte d'inclusion est violée les instants appartenant au domaine effectif qui ne sont pas présents dans le domaine d'observation sont ajoutés à ce dernier. Dans l'exemple que nous venons de considérer si l'instantané ⟨ 2002, 180 ⟩ est inséré dans l'historique effectif de la propriété IPV alors son domaine d'observation est élargi pour inclure l'année 2002.

#### Sélecteurs des caractéristiques des propriétés temporelles

Les opérateurs élémentaires pour accéder aux caractéristiques des propriétés temporelles aussi bien celles de transaction que celles de validité sont obtenirHistoEffectif et obtenirHistorique. Le premier permet de retrouver l'historique effectif d'une instance de propriété. Le second construit un historique à partir de l'historique effectif et du domaine d'observation en utilisant la modalité d'interpolation associée à la propriété en question. Dans le cas des propriétés de transaction le modalité d'interpolation est toujours celle en escalier.

Pour résumer nous donnons dans la figure 3.12 les spécifications en ODL des interfaces modélisant les différents types de propriétés temporelles.

### 3.2.2 Historisation des classes et des associations

#### Historisation des extensions de classes

Tout comme les propriétés les classes peuvent être fugitives ou temporelles. Dans le cas d'une classe fugitive seul le dernier état de son extension est pertinent vis-à-vis des

5. La fonction instant_présent retrouve l'instant associé à l'horloge du système.

```
interface InstanceDePropriété<T> {
  void modifierValeur(T o);
  T obtenirValeur();
}
interface InstanceDePropriétéTemporelle<T> :
                    InstanceDePropriété<T> {
  Historique<T> obtenirHistorique();
}
interface InstanceDePropriétéDeValidité<T> :
          InstanceDePropriétéTemporelle<T> {
  void modifierDomaineObservation(EDI s);
  void modifierHistoEffectif(Historique h);
}
```

```
interface InstanceDePropriétéDiscrète :
            InstanceDePropriétéDeValidité<T>;
interface InstanceDePropriétéEscalier :
            InstanceDePropriétéDeValidité<T>;
interface InstanceDePropriétéLinéaire :
            InstanceDePropriétéDeValidité<T>;
interface InstanceDePropriétéDeTransaction<T> :
            InstanceDePropriétéTemporelle<T> {
  void activer();
  void susprendre();
  enum { Activé, Désactivé } étatDActivation();
}
```

FIG. 3.12 – *Interfaces modélisant le comportement des propriétés temporelles.*

applications et donc stocké. A contrario, dans le cas d'une classe temporelle, on s'intéresse à observer son évolution au cours du temps, que ce soit par rapport au monde modélisé, ou par rapport à la vie de la base de données.

L'approche utilisée pour historiser les extensions des classes temporelles diffère radicalement de celle utilisée pour historiser les valeurs des propriétés temporelles. Dans le contexte des propriétés temporelles, l'historisation se fait au travers des concepts de domaine d'observation, d'historique effectif, et de modalité d'interpolation, caractéristiques qui permettent de dériver l'historique "complet" d'une instance de cette propriété. Il s'agit donc essentiellement d'une approche à base d'historiques. Dans le contexte des classes temporelles, l'historisation est fondée sur les concepts d'*extension complète* et de *domaine d'observation d'un objet*, concepts à partir desquels est défini celui d'*extension d'une classe à un instant donné*.

L'extension complète d'une classe temporelle est formée de tous les objets d'une classe ayant été créés (au moyen de l'opérateur new), même si par la suite ils sont *effacés* (au moyen de la méthode *delete*). À ce propos, une distinction est faite en TEMPOS entre l'*effacement* et la *destruction* d'un objet d'une classe temporelle. La destruction d'un objet entraîne sa suppression de l'extension complète de sa classe, alors que son effacement n'entraîne qu'une modification sur son domaine d'observation, modification qui éventuellement se réflète par son absence de l'extension de sa classe à certains instants. Concrètement, la destruction d'un objet temporel se fait au travers de la méthode destroy. Cette méthode ne s'applique pas aux objets temporels de transaction. En effet, détruire un objet temporel "de transaction" reviendrait à dire qu'un objet qui a existé dans la base de données n'y a jamais existé ! Bien entendu, des fonctionnalités pour effacer physiquement des informations sur le passé (au sens du temps de transaction) peuvent être fournis par un système donné, mais elles concernent le niveau "physique" et ne sont pas visibles au niveau "logique". Ainsi, en TSQL2 ce type d'opération

(dite de *nettoyage*) apparaît au niveau des outils d'administration du SGBD [Sno95b].

Le domaine d'observation d'un objet temporel est l'ensemble des instants auxquels l'information portée par cet objet est observée. À titre d'exemple, considérons une classe modélisant les types de produits assemblés par une usine. Si la classe est déclarée comme temporelle (que ce soit par rapport au temps de validité ou de transaction), le domaine d'observation d'un objet de cette classe peut par exemple dénoter la période de temps au cours de laquelle le type de produit dénoté par cet objet est assemblé par l'usine. Sous cette hypothèse, si la classe en question est "de transaction", le domaine d'observation d'un objet de cette classe dénote plus précisément l'ensemble des instants auxquels la base de données "sait" que le produit correspondant est assemblé, tandis que si la classe est "de validité", ce domaine dénote l'ensemble d'instants où ce produit est assemblé en réalité.

La valeur du domaine d'observation d'un objet temporel est accessible au travers d'une méthode associée aux classes temporelles, que nous appelons domaineDeValidité ou domaineDeTransaction, selon la dimension temporelle concernée. Dans les deux cas, le type du résultat de cette méthode est un EDI (Cf. figure 3.13).

```
interface ObjetTemporel;

interface ObjetDeValidité : ObjetTemporel {
  EDI domaineDeValidité();
  void modifierDomaineDeValidité(EDI);
}
```

```
interface ObjetDeTransaction : ObjetTemporel {
  EDI domaineDeTransaction();
  enum {Activé, Désactivé} étatDActivation();
  void activer();
  void suspendre();
};
```

FIG. 3.13 – *Interfaces modélisant le comportement des classes temporelles*

Les domaines d'observation des objets temporels évoluent de façon analogue aux domaines d'observation des propriétés temporelles (voir ci-haut). Ainsi, le domaine d'observation de chacun des objets d'une classe temporelle de transaction, évolue automatiquement avec l'horloge du système. Cette évolution "automatique" peut être contrôlée au travers des méthodes activer et suspendre définies au niveau de l'interface ObjetDeTransaction : la méthode suspendre a pour effet de susprendre l'évolution automatique du domaine de l'objet sur lequel elle s'applique, et la méthode activer a l'effet contraire. Par défaut, lorsqu'un objet est créé, l'évolution du domaine temporel est activée. Lorsqu'un objet est effacé (au travers de l'opérateur delete), l'évolution de son domaine d'observation est suspendue (mais l'objet n'est pas physiquement supprimé de la base).

S'agissant des classes temporelles de validité, le domaine d'observation des objets est fixé au travers d'une méthode modifierDomaineDeValidité, prenant en argument un EDI.

L'extension d'une classe temporelle C à un instant I est définie comme le sous-ensemble des objets de l'extension complète de C dont le domaine d'observation contient I. On remarque que

les valeurs de l'extension d'une classe temporelle à différents instants forment un historique de collections, chacune de ces collections correspondant à un état de l'extension de cette classe. En ce sens, on peut dire qu'une base de données comportant une classe temporelle "gère" l'historique de l'extension de cette classe.

### Historisation des associations binaires

Nous rappelons qu'en ODMG, une association binaire est modélisée à partir de deux *propriétés inverses* attachées aux classes participant à cette association. Le qualificatif "inverse" est là pour indiquer que les instances de ces propriétés vérifient les contraintes d'intégrité référentielles définies dans [CB97].

Suivant cette approche, nous modélisons les associations binaires temporelles au travers de deux *propriétés temporelles inverses* de même nature (c.à.d. même dimension et même modalité d'interpolation). À nouveau, l'adjectif "inverse" indique que les propriétés en question vérifient des contraintes d'intégrité référentielles que nous définissons ci-après.

**Definition 2** (*Propriétés temporelles inverses*). Deux propriétés temporelles P1 et P2 respectivement attachées aux classes C1 et C2 sont *inverses*, si les conditions décrites ci-après sont satisfaites. Dans cette description, la notation extension(C) dénote l'extension complète de la classe C et O.P dénote l'historique de l'instance de la propriété temporelle P associée à l'objet O. Les opérateurs Valeur et Domaine sont ceux définis dans le § 3.1.2.

- S'il s'agit d'une association 1-1 alors :

  $\forall$ O1 $\in$ extension(C1), $\forall$ i $\in$ Domaine(O1.P1)
  i $\in$ Domaine(Valeur(O1.P1, i).P2) $\wedge$ O1 = Valeur(Valeur(O1.P1, i).P2, i)
  $\wedge$ $\forall$ O2 $\in$ extension(C2), $\forall$ i $\in$ Domaine(O2.P2)
  i $\in$ Domaine(Valeur(O2.P2, i).P1) $\wedge$ O2 = Valeur(Valeur(O2.P2, i).P1, i)

- S'il s'agit d'une association 1-N où P1 modélise le rôle mono-valué et P2 celui multi-valué alors[6] :

  $\forall$ O1 $\in$ extension(C1), $\forall$ i $\in$ Domaine(O1.P1)
  i $\in$ Domaine(Valeur(O1.P1, i).P2) $\wedge$ O1 $\in$ Valeur(Valeur(O1.P1, i).P2, i)
  $\wedge$ ($\forall$ O2 $\in$ extension(C2), $\forall$ i $\in$ Domaine(O2.P2), $\forall$ O1 $\in$ Valeur(O2.P2, i)
  i $\in$ Domaine(O1.P1) $\wedge$ O2 = Valeur(O1.P1, i)

- S'il s'agit d'une association N-M alors :

  ($\forall$ O1 $\in$ extension(C1), $\forall$ i $\in$ Domaine(O1.P1), $\forall$ O2 $\in$ Valeur(O1.P1, i)
  i $\in$ Domaine(O2.P2) $\wedge$ O1 $\in$ Valeur(O2.P2, i)
  $\wedge$ ($\forall$ O2 $\in$ extension(C2), $\forall$ i $\in$ Domaine(O2.P2), $\forall$ O1 $\in$ Valeur(O2.P2, i)
  i $\in$ Domaine(O1.P1) $\wedge$ O2 $\in$ Valeur(O1.P1, i))

□

6. Le cas où P1 correspond au rôle multi-valué et P2 au rôle mono-valué s'obtient par symétrie.

On remarque que les contraintes contenues dans cette définition font intervenir le concept d'extension "complète" qui n'est défini que dans le cas des classes temporelles. De ce fait les classes participant à une association temporelle binaire doivent elles-mêmes être temporelles. A contrario, les attributs temporels (qui sont des propriétés mais ne participent pas à des associations) peuvent être définis même sur des classes fugitives.

### 3.2.3 Exemples récapitulatifs

Considérons une application concernant les cours suivis au cours du temps par des étudiants. Nous choisissons de modéliser les données de cette application au moyen d'une classe temporelle de transaction Étudiant possédant une propriété de transaction suit dont le type est set<Cours> (la classe Cours n'est pas décrite ici). Aussi bien la classe Étudiant que sa propriété suit sont observées à la granularité du jour. La domaine d'observation associé à un objet de la classe Étudiant dénote l'ensemble d'instants auxquels l'étudiant en question est inscrit à l'école. L'historique d'une instance de propriété de la classe Étudiant modélise quant à elle l'évolution de la connaissance de la base de données sur les cours suivis par un étudiant jour par jour au cours de sa scolarité. Le domaine d'observation d'une instance de cette propriété n'est pas convexe (il n'est pas défini pendant les périodes de vacances).

La table 3.1 décrit une suite de mises à jour dans le contexte de cette application et montre comment ces mises à jour peuvent être exprimées à partir des opérateurs décrits ci-dessus.

Considérons maintenant une classe temporelle "de validité" UnitéDeProduction munie d'une propriété de validité nommée superviseur. Cette propriété évolue en escalier à la granularité du jour et son historique prend des valeurs de type Superviseur. Elle est l'un des chemins de traversée d'une association temporelle 1-1 dont le chemin inverse est une autre propriété temporelle Superviseur::supervise. Le domaine d'observation d'un objet de cette classe modélise l'ensemble d'instants auxquels l'unité en question existe dans l'usine, alors que le domaine d'observation d'une instance de la propriété UnitéDeProduction::superviseur modélise la période de temps pendant laquelle une unité est en fonctionnement.

La table 3.2 montre un scénario d'évolution pour un objet de cette classe. La notation [i..] (resp. [..i]) dénote l'intervalle contenant tous les instants de même granularité que i et supérieurs ou égaux (resp. inférieurs ou égaux) à lui. Étant donnés une propriété P et un objet O, O.P dénote l'instance de la propriété P attachée à l'objet O. Cette instance de propriété est traitée comme un objet auquel il est possible d'appliquer des méthodes. Ainsi, O.P.obtenirHistorique() dénote l'historique de l'instance de propriété O.P. On remarque que ces notations sont incompatibles avec le modèle de l'ODMG où les instances de propriété ne sont pas traitées comme des objets. Nous reviendrons sur ce point dans le § 3.3.3, où nous proposerons des notations alternatives pour ces opérations.

| | |
|---|---|
| **Date** | 1/9/98 |
| **Événement** | Un nouvel étudiant s'inscrit |
| **Opération** | S = new Étudiant |
| **Résultat** | S.étatDActivation() = Activé ; S.Suit.étatDActivation() = Activé<br>S.Suit.obtenirHistorique() = {⟨[1/9/98..1/9/98], { }⟩} |
| **Date** | 3/9/98 |
| **Événement** | L'étudiant s'inscrit dans les modules de Maths et de Chimie |
| **Opération** | S.Suit.modifierValeur({Maths, Chimie}) |
| **Résultat** | S.domaineDeTransaction() = [1/9/98..3/9/98];<br>S.Suit.obtenirHistorique() = {⟨[1/9/98..2/9/98], { }⟩, ⟨[3/9/98..3/9/98], {Maths, Chimie}⟩} |
| **Date** | 8/10/98 |
| **Événement** | L'étudiant se retire de l'école |
| **Opération** | S.suspendre() |
| **Résultat** | S.étatDActivation() = Désactivé; S.Suit.étatDActivation() = Désactivé;<br>S.domaineDeTransaction() = [1/9/98..7/10/98]; S.Suit.obtenirHistorique() =<br>{⟨[1/9/98..2/9/98], {}⟩, ⟨[3/9/98..7/10/98], {Maths, Chimie}⟩} |
| **Date** | 1/2/99 |
| **Événement** | L'étudiant se réinscrit; il prend les modules de Logique et de Bases de Données |
| **Opération** | S.activer(); S.Suit.modifierValeur({Logique, Bases de Données}) |
| **Résultat** | S.étatDActivation() = Activé ; S.Suit.étatDActivation() = Activé;<br>S.domaineDeTransaction() = {[1/9/98..3/9/98], [1/2/99..1/2/99]};<br>S.Suit.obtenirHistorique() = {⟨[1/9/98..2/9/98], {}⟩, ⟨[3/9/98..8/10/98], {Maths, Chimie}⟩, ⟨[1/2/99..1/2/99], {Logique, Bases de Données}⟩} |

TAB. 3.1 – *Scénario d'évolution d'un objet suivant le temps de transaction*

### 3.2.4 TempODL : un langage de définition de schémas temporels

Dans le but de rendre facilement accessibles aux développeurs d'applications les concepts de propriété et de classe temporelles définis ci-dessus, nous proposons une extension simple du langage de définition de schémas ODL, que nous baptisons TEMPODL.

Les seules extensions à ODL apportées par TEMPODL concernent la déclaration de classes, d'attributs et d'associations binaires. Dans la grammaire BNF de ODL [CB97], les terminaux correspondant à ces trois types de déclarations sont notés respectivement `class_header`, `attr_decl` et `rel_decl`. Nous transcrivons partiellement les règles les concernant ci-après :

```
class_header::= class <identifier> extends <scopedName> ...
<attr_decl>::= [readonly] attribute <domain_type> <identifier>
<rel_decl>::= relationship <class_name> <identifier> inverse
                                          <identifier> ::<identifier>
<template_type_spec> ::= array_type | string_type | coll_type
```

En TempODL, ces quatre règles sont remplacées par celles données ci-dessous. On peut facilement vérifier que ces nouvelles règles subsument celles qu'elles remplacent, ce qui entraîne

| | |
|---|---|
| **Événement** | Une nouvelle unité de production est introduite le 1/4/98, et mise en fonctionnement sous la supervision de S1. |
| **Opération** | U = new UnitéDeProduction;<br>U.superviseur.modifierDomaineObservation([1/4/98..]);<br>U.superviseur.modifierHistoEffectif({⟨1/4/98, S1⟩}) |
| **Résultat** | U.superviseur.obtenirHistorique() = {⟨[1/4/98..], S1⟩}<br>∀ i ∈ [1/4/98..] ( S1.supervise.obtenirHistorique().Valeur(i) = U ) |
| **Événement** | À partir du 1/6/98, le superviseur de l'unité U est S2 |
| **Opération** | U.superviseur.modifierHistoEffectif(U.superviseur.obtenirHistoEffectif()<br>$\cup_+$ {⟨1/6/98, S2⟩}) |
| **Résultat** | U.superviseur.obtenirHistorique() = {⟨[1/4/98..31/5/98], S1⟩, ⟨[1/6/98..], S2⟩}<br>∀ i ∈ [1/6/98..] ( S2.supervise.obtenirHistorique().Valeur(i) = U )<br>∀ i ∈ [1/6/98..] ( S1.supervise.obtenirHistorique().Valeur(i) = NULL ) |
| **Événement** | L'unité est sortie de l'usine à partir du 6/8/98. |
| **Opération** | U.modifierDomaineDeValidité(U.domaineDeValidité() ∩ [..5/8/98]) |
| **Résultat** | U.domaineDeValidité() = [1/4/98..5/8/98]<br>U.superviseur.obtenirHistorique() = {⟨[1/4/98..31/5/98], S1⟩,<br>⟨[1/6/98..5/8/98], S2⟩}<br>∀ i ∈ [6/8/98..] ( S2.supervise.obtenirHistorique().Valeur(i) = NULL ) |
| **Événement** | L'unité est ré-introduite à l'usine à partir du 1/1/99 mais elle n'est pas mise en service. |
| **Opération** | U.modifierDomaineDeValidité(U.domaineDeValidité ∪ [1/1/99..]) |
| **Résultat** | U.domaineDeValidité() = {[1/4/98..5/8/98], [1/1/99..]}<br>U.superviseur.obtenirHistorique() inchangé |
| **Événement** | L'unité est remise en service avec pour superviseur S3 pour la période allant du 1/2/99 au 31/3/99. |
| **Opération** | U.superviseur.modifierDomaineObservation(U.superviseur.domaineDeValidité()<br>∪ [1/2/99..31/3/99])<br>U.superviseur.modifierHistoEffectif(U.superviseur.obtenirHistoEffectif()<br>$\cup_+$ {⟨1/2/99, S3⟩}) |
| **Résultat** | U.domaineDeValidité() inchangé<br>U.superviseur.obtenirHistorique() = {⟨[1/4/98..31/5/98], S1⟩,<br>⟨[1/6/98..5/8/98], S2⟩, ⟨[1/2/99..31/3/99], S3⟩}<br>∀ i ∈ [1/2/99..31/3/99] ( S3.supervise.obtenirHistorique().Valeur(i) = U ) |

TAB. 3.2 – *Scénario d'évolution d'un objet suivant le temps de validité*

que TEMPODL est une extension conservatrice d'ODL.

<class_header>::= [<temporal_dimension> <granularity>] **class** ...
<attr_decl>::= [<temporal_dimension> <assumption> <granularity>] [**readonly**]...
<rel_decl>::= [<temporal_dimension> <assumption> <granularity>] **relationship**...
<temporal_dimension>::= **valid** | **transaction**
<assumption>::= **discrete** | **stepwise** | **linear**[7]
<granularity> ::= <identifier>

<template_type_spec> ::= array_type | string_type | coll_type | history_type
history_type ::= **history**'<' <simple_type_spec> '>'

À titre d'exemple, nous décrivons le schéma de notre base de données exemple dans la notation TEMPODL. Les propriétés temporelles sont observées à la granularité du jour. Les attributs production et qualité sont discrets, puisque modélisant des occurrences d'événements ( "la production aujourd'hui a été de tant" ), et non pas des états ( "le salaire de tel employé est actuellement de tant" ). Les classes UnitéDeProduction, Employé et les sous-classes de cette dernière sont temporelles, car elles participent à des associations temporelles.

```
valid granularity Day class UnitéDeProduction(extent LesUnités, key codeUnité) {
  attribute string codeUnité;
  valid stepwise Day relationship Superviseur superviseur inverse Superviseur::supervise;
  valid stepwise Day relationship set<Ouvrier> ouvriers inverse Ouvrier::unité;
  valid discrete Day attribute short production;
  valid discrete Day attribute float qualité;
}
valid granularity Day class Employé (extent LesEmployés, key (nom, dateNaissance)) {
  attribute string nom;    attribute Instant dateNaissance;
  valid stepwise Day attribute float salaire;
}
valid granularity Day class Ouvrier extends Employé (extent LesOuvriers) {
  valid stepwise Day relationship UnitéDeProduction unité inverse UnitéDeProduction::ouvriers;
}
valid granularity Day class Superviseur extends Employé (extent LesSuperviseurs) {
  valid stepwise Day relationship UniteDeProduction supervise
                                  inverse UnitéDeProduction::superviseur;
}
```

7. Lorsqu'un attribut est linéaire, le type de l'attribut doit être numérique.

Dans cet exemple, toutes les propriétés participant aux clés des extensions de classe sont fugitives. Lorsqu'une propriété temporelle intervient dans une clé, on parle d'une *clé temporelle*. Étant données une classe possédant une clé temporelle, et un instant $I$, les valeurs de cette clé à l'instant $I$ doivent différer pour tout couple d'objets de cette classe. La valeur d'une clé temporelle pour un objet donné à un instant $I$, est obtenue par juxtaposition des valeurs prises à l'instant $I$, par chacune des propriétés composant cette clé, en convenant que la valeur d'une propriété fugitive est la même à tout instant. Dans le cas où pour un objet donné, l'une des propriétés temporelles participant à une clé est indéfinie à un instant donné, on considère que la valeur de la clé vis-à-vis de cet objet à cet instant, diffère de celle de tous les autres objets de son extension. La sémantique adoptée est alors celle qu'on obtiendrait à partir du prédicat SQL2 UNIQUE[8], en convenant qu'une propriété temporelle prend la valeur NULL aux instants sortant de son domaine temporel. On rappelle que dans la sémantique du prédicat UNIQUE de SQL2, le "bénéfice du doute" est accordé aux valeurs nulles. Autrement dit, les n-uplets (a, b), (a, NULL) et (NULL, NULL) sont considérés comme n'étant pas égaux [Cel95], et on peut même avoir deux occurrences du couple (NULL, NULL) sans pour autant violer la condition.

## 3.3 Migration d'applications vers des BD temporelles

Dans l'introduction de cette thèse, nous avons fait référence aux notions de "compatibilité ascendante" et de "migration transparente d'applications vers des BD temporelles", que nous avons illustrées au travers d'un exemple. Dans cette section, nous formalisons ces notions, et introduisons les mécanismes du modèle TEMPOS permettant d'assurer ces propriétés. Cette formalisation s'inspire partiellement de celle développée par [BBJS97, SBJS98] dans le cadre des extensions temporelles de SQL. Une version préliminaire de ce travail est décrite dans [DFS99b].

### 3.3.1 Définitions

Dans l'optique de formaliser les concepts de compatibilité et de transition entre modèles de données, nous adoptons la définition suivante.

**Definition 3** (*Modèle de données*). Un modèle de données M est un n-uplet $(\mathsf{D}, \mathsf{Q}, \mathsf{U}, [\![\,]\!]_M)$ composé d'un ensemble d'instances de bases de données D, d'un ensemble de requêtes d'interrogation Q, d'un ensemble de requêtes de mise à jour U, et d'une fonction d'évaluation $[\![\,]\!]_M$. Étant données une requête de mise à jour $\mathsf{u} \in \mathsf{U}$, une requête d'interrogation $\mathsf{q} \in \mathsf{Q}$ et

8. Étant donné que le standard ODMG reste très flou en ce qui concerne le traitement des valeurs nulles, nous adoptons la sémantique qu'il leur est accordée en SQL2

une instance de base de données $\mathsf{db} \in \mathsf{D}\Gamma[\![\, \mathsf{u(db)} \,]\!]_M$ dénote une instance de base de donnéesΓ et $[\![\, \mathsf{q(db)} \,]\!]_M$ une instance d'une structure de données. □

AinsiΓune base de données est vue comme une entité abstraiteΓà laquelle il est possible d'appliquer des mises à jour (opérateurs permettant d'obtenir une base de données à partir d'une autre)Γet des requêtes d'interrogation.

Dans le cadre des SGBD conformes à l'ODMGΓl'ensemble de requêtes d'interrogation Q ne se restreint pas à celles reconnues par l'interpréteur OQL : il inclut aussi les opérations de navigation et d'accès à la valeur d'un attributΓeffectuées depuis un langage de programmation au travers de la passerelle correspondante. Les requêtes de mise à jour quant à ellesΓincluent les créations et suppressions d'objetsΓet les mises à jour des propriétésΓopérations effectuées toujours depuis un langage de programmation.

À partir de la définition ci-dessusΓil est relativement facile de formaliser la notion de compatibilité ascendante entre modèles de données. Cette notion définit en particulier les contraintes nécessaires à la migration transparente des applicationsΓlors d'un basculement d'un SGBD classique vers un SGBD temporel.

**Definition 4** (*Compatibilité ascendante*). Un modèle de données $\mathsf{M'} = (\mathsf{D'},\mathsf{Q'},\mathsf{U'},[\![\,]\!]_{M'})$ est compatible vers le haut vis-à-vis du modèle $\mathsf{M} = (\mathsf{D},\mathsf{Q},\mathsf{U},[\![\,]\!]_M)$[9] ssi :

- $\mathsf{D} \subseteq \mathsf{D'}\Gamma\mathsf{Q} \subseteq \mathsf{Q'}$ et $\mathsf{U} \subseteq \mathsf{U'}$ (condition dite de "compatibilité syntaxique ascendante" dans [BBJS97]).
- Pour tout db dans DΓpour tout q dans QΓpour toute suite $\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_n$ de requêtes de mise à jour dans UΓet pour toute suite d'instants $\mathsf{d}_1, \ldots \mathsf{d}_n, \mathsf{d}_{n+1}$ :
$$[\![\, \mathsf{q}^{\mathsf{dn+1}}(\mathsf{u}_\mathsf{n}^{\mathsf{dn}}( \ldots \mathsf{u}_2^{\mathsf{d2}}( \mathsf{u}_1^{\mathsf{d1}}(\mathsf{db})))) \,]\!]_M = [\![\, \mathsf{q}^{\mathsf{dn+1}}(\mathsf{u}_\mathsf{n}^{\mathsf{dn}}( \ldots \mathsf{u}_2^{\mathsf{d2}}(\mathsf{u}_1^{\mathsf{d1}}(\mathsf{db})))) \,]\!]_{M'}$$

□

On remarque que dans cette dernière expressionΓles requêtes de mise à jour et d'interrogation sont paramétrées par des instants. En effetΓdans beaucoup de modèles de données temporelsΓla sémantique des accès à la base dépend de l'instant associé à la transaction dans laquelle ils opèrent [CDI+97].

TEMPOS est une extension conservatrice du modèle d'objets de l'ODMGΓpuisque les types qu'il fournit ne se substituent pas à ceux déjà présents dans le modèle de l'ODMGΓ mais viennent s'ajouter à ces derniers.

La notion de *modèle temporel bi-accessible* dont nous avons discuté dans l'introduction (Cf. § 1.2.2)Γse formalise au travers d'un couple de modèles vérifiant certaines propriétés.

**Definition 5** (*Modèle de données temporelles bi-accessible*). Une paire de modèles de données $(\mathsf{M_S},\mathsf{M_T})$ tels que $\mathsf{M_S} = (\mathsf{D_S}, \mathsf{Q_S}, \mathsf{U_S}, [\![\,]\!]_{M_S})$ et $\mathsf{M_T} = (\mathsf{D_T}, \mathsf{Q_T}, \mathsf{U_T}, [\![\,]\!]_{M_T})$ forment un modèle

9. On dira aussi que M' est une *extension conservatrice* de M

de données temporelles bi-accessible ssi :

- $M_T$ est une extension conservatrice de $M_S$.
- pour tout $db_s \in D_S$ il existe un $db_t \in D_T$ tel que pour tout $q \in Q_S$ pour toute suite de mises à jour $u_1, u_2, \ldots, u_n \in U_S$ et pour toute suite d'instants $d_1, \ldots d_n, d_{n+1}$:
  $[\![ q^{dn+1}(u_n^{dn}( \ldots u_2^{d2}(u_1^{d1}(db_s))))]\!]_{M_S} = [\![ q^{dn+1}(u_n^{dn}( \ldots u_2^{d2}(u_1^{d1}(db_t))))]\!]_{M_T}$

□

Par rapport à la figure 1.2 page 9, $db_s$ correspond à la base de données à gauche de la figure, tandis que $db_t$ correspond à celle de droite. La vue fugitive définie au dessus de $db_t$ se comporte de la même façon que $db_s$ (Cf. figure 3.14), alors que la vue temporelle permet d'accéder à tout le contenu de $db_t$. Le processus de transformation de $db_s$ en $db_t$ est souvent appelé *adaptation* ou encore *conversion des instances* dans la terminologie sur l'évolution de schéma [BF99]. Ce point fait l'objet de l'alinéa suivant.

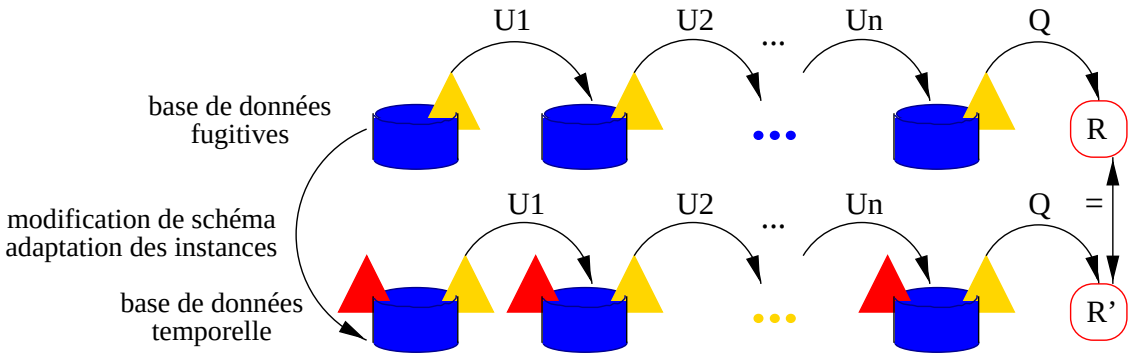



FIG. 3.14 – *La vue fugitive d'une base de données temporelles bi-accessible se comporte de la même manière que la base de données fugitive à partir de laquelle elle est issue. Dans ce dessin,* U1, U2, *etc. sont des mises à jour "non-temporelles",* Q *est une requête non-temporelle,* R *et* R' *sont des instances de structures de données. Les triangles jaunes (clairs) dénotent des schémas fugitifs alors que les triangles rouges (foncés) correpondent à des schémas temporels.*

TEMPOS fournit des mécanismes assurant la bi-accessibilité des bases de données temporelles, facilitant ainsi la migration d'applications opérant sur une base de donnée fugitive lorsqu'une modification de schéma vient ajouter une dimension temporelle à une ou plusieurs composantes de son schéma (par "composante" on entend ici une classe ou une propriété). Dans le cas des propriétés temporelles, ces mécanismes ne s'appliquent que lorsqu'une propriété fugitive est remplacée par une propriété temporelle évoluant en escalier. En effet, l'idée de base de la bi-accessibilité est que lorsque la valeur d'une propriété temporelle est modifiée par une application non-temporelle, la valeur affectée par cette mise à jour reste la valeur "courante" de la propriété jusqu'à la prochaine modification. Or, cette caractéristique est inhérente à la modalité d'interpolation en escalier. Par ailleurs, le remplacement d'une pro-

priété fugitive par une propriété discrète ou linéaire implique un changement radical de la sémantique de la propriété et il est donc normal que les répercussions de cette modification de schéma soient prises en compte au cas par cas au niveau applicatif.

### 3.3.2 Adaptation des instances

L'algorithme ci-dessous décrit comment les objets d'une base de données sont adaptés lors d'une modification de schéma dans laquelle des classes et/ou des propriétés fugitives sont remplacées par des classes et des propriétés temporelles. Conceptuellement cet algorithme doit être appliqué au moment de la modification du schéma. Cependant dans le cas de bases de données volumineuses il est envisageable d'employer des techniques d'évaluation paresseuse afin de retarder cette exécution jusqu'au moment où l'un des objets concernés est accédé par un programme d'application.

**Algorithme:** *Opérateur d'adaptation d'instances*

**Données et variables locales:**

```
classes_modifiées: Set<Classe>; /* classes passant du statut fugitif au statut temporel */
propriétés_modifiées: Set<Propriété>; /* propriétés passant du statut fugitif au statut temporel */
O: Object; /* objet à convertir */
copie_O: Object; /* variable utilisée pour stocker une copie de l'objet O */
```

**Partie procédurale:**

```
copie_O := O.copy(); /* une copie de O avec son ancienne structure est stockée temporairement. */
Adapter la structure de l'objet O au nouveau schéma, tout en conservant son identificateur;
si classe(O) dans classes_modifiées alors
  si (dimensionTemporelle(classe(O)) = "temps de transaction") alors
     O.activer() /* le domaine d'observation de l'objet doit évoluer automatiquement */
  sinon
     O.modifierDomaineDeTransaction([instant_courant()..])
pour chaque p dans propriétés(classe(O))
    si (p dans propriétés_modifiées) {
       si (dimensionTemporelle(p) = "temps de transaction") alors
          O.p.activer(); /* le domaine d'observation de l'instance de propriété doit évoluer automa-
                            tiquement */
       sinon si (modalitéDInterpolation(p) = "en escalier") alors
          O.p.modifierDomaineObservation([instant_courant()..])
       O.p.modifierValeur(copie_O.p.obtenirValeur())
    }
    sinon O.p = O_copy.p /* la propriété reste fugitive */
```

### 3.3.3 Manipulation de propriétés temporelles et modes d'accès

En règle généraleΓles langages de programmation à objets offrent seulementΓun opérateur pour accéder à la valeur d'une propriétéΓet un autre pour modifier cette valeur. Par exempleΓ en JavaΓl'accès à la valeur d'une propriété s'effectue au travers de l'opérateur "."Γalors que les mises à jour s'expriment à partir de l'opérateur d'affectation O.P = ... .

De son côtéΓl'extension du modèle de l'ODMG que nous proposons introduit une spécialisation du type "propriété"Γà savoir "propriété temporelle"Γmuni d'un jeu d'opérateurs beaucoup plus riche. Dans la suiteΓnous proposons des notations pour invoquer ces opérateurs depuis les programmes d'applicationΓen respectant le principe du modèle à objets de l'ODMGΓet en particulier le fait que les instances de propriétés ne sont pas traitées comme des entités à part entière. En d'autres termesΓil s'agit de faire en sorte qu'aucune expression dans un programme ou dans une requête ne soit de type "instance de propriété"Γce qui n'est pas le cas dans la notation que nous avons utilisée dans les tables 3.1 et 3.2 précédemment.

Conformément à l'objectif de fournir un modèle bi-accessibleΓles opérateurs "." et O.P = ... ont des sémantiques différentes selon l'application depuis laquelle ils sont invoqués. Pour modéliser cette situationΓnous introduisons le concept de *mode d'accès*Γcaractéristique attachée à chaque applicationΓet pouvant prendre l'une des deux valeurs :

- **Fugitif**. Dans ce casΓles sémantiques des notations "." et de O.P = ... lorsque P est une propriété temporelleΓsont définies par les opérateurs obtenirValeur et modifierValeur respectivement. De plusΓtoute référence à l'extension d'une classe temporelle dans un programme ou dans une requêteΓretrouve l'extension observée à l'instant courant (celui associé au début de la transaction dans laquelle opère le programme).
- **Temporel**. Les notations "." et O.P = ... lorsque P est une propriété temporelleΓ correspondent aux opérateurs obtenirHistorique et modifierHistorique. Toute référence à l'extension d'une classe temporelle dans un programme ou dans une requêteΓretrouve l'extension complète de cette classe.

AinsiΓdans le mode fugitifΓles applications perçoivent les propriétés temporelles comme étant fugitivesΓleur valeur étant celle que prend leur historique à l'instant attaché à la transaction en cours. A contrarioΓdans le mode temporelΓles propriétés temporelles sont perçues comme étant de type historique. À titre d'exempleΓconsidérons la requête suivante sur notre base de données exemple :

```
element(select u.superviseur
        from LesUnités
        where u.codeUnité = "X")
```

Si l'application qui soumet cette requête est en mode fugitifΓle type du résultat est Superviseur. Par contreΓsi son mode est temporelΓle type du résultat est Historique<Superviseur>.

Ceci est valable aussi bien lorsque superviseur est une propriété de validité (évoluant en escalier)Γqu'une propriété de transaction.

Le concept de mode d'accèsΓpermet de simuler un mécanisme de vuesΓrépondant aux besoins de bi-accessibilité spécifiés ci-dessus. De plusΓen prenant le mode fugitif comme étant le mode par défautΓTempos rend possible la migration transparente des applications opérant sur une base de données fugitivesΓlorsque son schéma devient "temporel".

Pour ce qui est des autres opérateurs sur les propriétés temporellesΓla notation utilisée pour leur invocation est obtenue par concaténation du nom de l'opérateur est de celui de la propriété. AinsiΓpour modifier le domaine d'observation de la propriété temporelle salaire pour un employé EΓon écrit E.modifierDomaineObservation_salaire( ... ).

## Récapitulatif

Le modèle de données Tempos est une extension conservatrice du modèle de données à objets du standard ODMG. Il est essentiellement composé d'un *modèle du temps*Γd'un *modèle d'historiques*Γet d'un *modèle de classes et de propriétés temporelles*.

Au regard du système de typesΓles deux premiers de ces modèles étendent la hiérarchie de types prédéfinis de l'ODMGΓen y introduisant des types dits "temporels". Modulo les problèmes induits par l'éventuelle absence de classes paramétréesΓl'implantation de ces modèles au dessus d'un système conforme à l'ODMG est en principe possible sans avoir à intervenir au niveau de son noyau. A contrarioΓle troisième des modèles ci-dessus touche directement aux concepts de base de l'ODMGΓà savoir ceux de classe et de propriété. Son intégration au sein d'un système existant est de ce fait beaucoup plus délicate.

Face à cette situationΓnous avons soigneusement séparé les concepts composant chacun de ces modèlesΓdéfinissant ainsi un système de types à trois niveaux. Le premier niveau comporte uniquement les types fournis par le modèle du temps; le deuxième rajoute ceux du modèle d'historiques; et le troisième intègre les concepts de classe et de propriété temporelles. L'intérêt de cette structuration est de permettre l'implantation de Tempos à divers niveaux de conformitéΓselon les besoins des applications ciblées et le degré d'extensibilité du système sous-jacent.

Le fait que Tempos supporte tous les concepts et fonctionnalités existants dans l'ODMGΓ rend possible la migration transparente de données stockées dans un SGBD conforme à l'ODMG vers un SGBD implantant Tempos. Dans le même espritΓTempos fournit des mécanismes permettant la migration transparente d'applications opérant sur une base de données fugitivesΓvers une base de données temporelles. Ceci constitue l'une des originalités majeures de Tempos par rapport aux autres extensions temporelles de l'ODMG (Cf. § 2.3.3 et § 1.2.2).

# Chapitre 4

# Interrogation et visualisation de bases d'objets temporels

Sur la base du modèle décrit dans le chapitre précédent, la plate-forme TEMPOS fournit deux interfaces pour l'interrogation et la visualisation de bases d'objets temporels. La conjonction de ces interfaces donne lieu à un outil permettant de formuler des requêtes temporelles de façon déclarative, et de construire des interfaces pour l'exploration des objets extraits par ces requêtes, selon un paradigme dans lequel le temps est traité comme une dimension à part entière.

## 4.1 Le langage de requêtes TempOQL

Implantés au sein de classes, les opérateurs du modèle TEMPOS fournissent certes une bonne approche pour formuler des requêtes portant sur des objets temporels, mais cela au prix d'une syntaxe malcommode, suivant un paradigme peu déclaratif et a fortiori difficile à optimiser. À titre d'exemple considérons la requête suivante portant sur le schéma de la figure 3.9 (page 53).

**Q5** : *Pour chaque employé, donner l'ensemble d'instants auxquels cet employé gagne plus de ce qu'il gagne le 1/2/99 ?*

```
/* type du résultat : bag<struct<employé: Employé, instants: EDI>> */
select employé: e,
       instants: e.salaire.Rsi("$$ > $1", list(e.salaire.Valeur(Instant("1/2/99")))).Domaine
                 /* Rsi est la méthode correspondant à l'opérateur Σ_im (Cf. annexe A). Son premier
                    argument est une requête décrite par une chaîne de caractères dans laquelle $$
                    dénote une valeur de l'historique, et $1, $2, etc. sont des paramètres. Les valeurs
                    effectives de ces paramètres sont données par le deuxième argument de Rsi. */
from LesEmployés as e
```

Cet exemple met en évidence au moins trois inconvénients de cette approche :

- **Syntaxe intriquée.** Imposé par les limitations du système de types de l'ODMG, le codage des opérateurs d'ordre supérieur au travers de méthodes prenant en paramètre une chaîne de caractères, donne lieu à une syntaxe peu lisible. Dans le pire des cas, ceci conduit à des emboîtements de guillemets, notamment lorsqu'à l'intérieur d'une fonction décrite au travers d'une chaîne de caractères, on veut faire référence à une autre constante de type chaîne de caractères.
- **Typage statique imprécis.** Toujours à cause des limitations du système de types de l'ODMG, le code des méthodes correspondant aux opérateurs d'ordre supérieur sur les historiques n'est pas typé de façon complètement statique. Ainsi, la méthode Rsi prend en deuxième paramètre une liste d'objets, dont le type exact dépend du contenu de la chaîne de caractères donnée en premier argument. Le contenu de cette chaîne n'étant pas connu au moment de la compilation du corps de la méthode Rsi, on est amené à déclarer statiquement que le type de son deuxième paramètre est List<Object> (Cf. annexe A), et à effectuer une partie de la vérification de type au moment de l'exécution de la méthode.
- **Impact négatif sur les possibilités d'optimisation.** Comme nous l'avons signalé dans le § 2.1.3, il n'existe pratiquement aucune technique permettant d'optimiser de façon "globale" les requêtes OQL comportant des appels de méthodes. Dans les interpréteurs OQL existants, les méthodes sont vues comme des boîtes noires auxquelles le contrôle du flot d'exécution est octroyé à certains points du plan d'évaluation d'une requête. Dans l'exemple ci-dessus, ceci entraîne que les traitements effectués par les appels aux méthodes Rsi, Valeur et Domaine, ne sont pas optimisés au même titre que les traitements effectués directement par l'interpréteur OQL.

Estimant que ces inconvénients rendent l'approche ci-dessus peu viable, nous proposons comme alternative une extension du langage OQL, baptisée TempOQL [FDS99].

En TempOQL les types du modèle Tempos sont primitifs, au même titre que les entiers, les réels, etc. La manipulation des instances de ces types se fait au travers de constructions syntaxiques prédéfinies et non pas au travers d'appels de méthodes. De plus, lorsque leur sémantique le permet, les opérateurs prédéfinis d'OQL sont étendus aux types temporels.

De par sa conception, TempOQL préserve la nature fonctionnelle d'OQL : toutes les constructions qu'il introduit sont des opérateurs, qu'il est possible de composer avec tous les autres opérateurs du langage (modulo les contraintes induites par les règles de typage).

### 4.1.1 Manipulation de valeurs temporelles

L'expression de constantes de type instant en TempOQL, se fait au travers de deux opérateurs dénotés au travers du symbole @. Utilisée dans une position d'infixe, @ prend en

argument un entier et une granularité, et construit un instant en interprétant l'entier comme la position d'un grain. Par exemple 31904 @ Jour dénote un instant à la granularité du jour. Dans sa version préfixée, @ prend en argument une chaîne de caractères, qui est interprétée en un instant par rapport à un système de formats par défaut, qui constitue de ce fait un paramètre du langage. En supposant que le système de formats par défaut contient le format européen des dates, la requête @ "16/3/1999" dénote un instant à la granularité du jour.

De façon analogue, les durées sont construites au travers de deux opérateurs notés # : l'un avec une position d'infixe, et l'autre avec une notation préfixée. En tant qu'infixe, # construit une durée à partir d'un entier et d'une granularité. Dans sa version préfixée, il construit une durée à partir d'une chaîne de caractères reconnue par le système de formats par défaut. En faisant certaines hypothèses sur la composition du système de formats par défaut, les expressions 2 # jour et # "2 jours" dénotent toutes les deux la même durée.

Il est par ailleurs possible de construire des instants à partir d'une granularité, au travers des opérateurs beginning, now et forever. L'opération now(G) retrouve l'instant de granularité G associé à l'horloge du système. Bien entendu, G doit partitionner la même ligne de temps que la granularité associée à l'horloge du système, faute de quoi l'approximation de l'instant indiqué par l'horloge vers la granularité G ne serait pas définie. L'opération beginning(G) (resp. forever(G)) quant à elle, retrouve l'instant correspondant au plus petit (resp. plus grand) grain de G, et ce quel que soit la ligne de temps sur laquelle est définie G.

Les opérateurs arithmétiques et de comparaison d'OQL sont étendus aux instants et aux durées lorsque les opérations respectives sont définies dans le modèle du temps (Cf. § 3.1.1). Par exemple, si $i_1$ et $i_2$ sont des requêtes de type instant et d est une requête de type durée, alors $i_1 < i_2$ est une requête de type booléen, $i_1 - i_2$ est une requête de type durée et $i_1 + d$ est une requête de type instant. La sémantique de ces opérateurs est décrite in extenso dans [Can97].

La conversion entre instants s'effectue au travers des opérateurs approx et expand, tous les deux prenant comme arguments un instant et une granularité (Cf. § 3.1.1). Du point de vue du typage, le résultat de approx(I, G) est un instant, alors que celui de expand(I, G) est un intervalle. La conversion de durées s'exprime au travers de l'opérateur convert, qui prend comme arguments une durée et une granularité (régulière par rapport à la granularité de la durée), et donne en résultat une durée de granularité G.

En ce qui concerne les ensembles d'instants et les intervalles, nous décrivons ci-après les constructeurs associés. Dans cette énumération, i, $i_1$, $i_2$, ... $i_n$ sont des requêtes de type instant, d est une requête de type durée et $x_1$, $x_2$, ... $x_n$ sont de type intervalle.

- Constructeurs d'intervalles
  - $[i_1 .. i_2]$ dénote l'intervalle de bornes $i_1$ et $i_2$. Si $i_1 > i_2$, il s'agit de l'intervalle vide.

  - [i | d] est équivalent à [i .. i + d]
  - [i .. ] est équivalent à [i .. forever(g)] où g est la granularité de i.
  - [ .. i] est équivalent à [beginning(g) .. i] où g est la granularité de i.

- Constructeurs d'EDI
  - iset($i_1$, $i_2$, ... $i_n$) est l'EDI contenant les instants $i_1$, $i_2$, ... $i_n$.
  - iset($x_1$, $x_2$, ... $x_n$) est l'EDI obtenu par union des intervalles $x_1$, $x_2$, ... $x_n$.

Trois sélecteurs sont définis sur le type EDI à savoir min et max et duration (Cf. § 3.1.1 page 46). Les deux premiers sont des extensions "naturelles" des opérateurs OQL sur les collections. On remarque que lorsque min (resp. max) est appliqué à un intervalle il retrouve sa borne inférieure (resp. supérieure). Ainsi il n'est pas nécessaire d'introduire des nouveaux mots clés dans le langage pour dénoter ces opérations comme c'est le cas dans TOOBIS [TOO96b] et TSQL2 [Sno95b] où begin et end sont employés à cet effet.

Les opérateurs ensemblistes d'OQL sont aussi étendus aux ensembles d'instants et a fortiori aux intervalles. Le type du résultat de ces opérateurs est EDI à l'exception de l'opérateur d'intersection de deux intervalles dont le type du résultat est Intervalle.

Une dernière famille d'opérateurs sur les intervalles concerne les relations binaires de comparaison. Suivant l'approche utilisée en TSQL2 nous ne retenons que quatre opérateurs de cette catégorie : <, =, overlaps (intersection) et meets le but étant de limiter le nombre de mots clés. Par rapport aux treize relations d'Allen [All83] ce jeu d'opérateurs n'est pas complet. Cependant toutes les relations d'Allen s'expriment par composition des opérateurs min et max avec les opérateurs de comparaison d'instants [Sno95b]. À titre d'exemple la relation *during* du système de relations d'Allen s'écrit $\mathsf{min}(x_1) > \mathsf{min}(x_2)$ and $\mathsf{max}(x_1) < \mathsf{max}(x_2)$. Par ailleurs l'opérateur meets en TempOQL possède une sémantique légèrement différente de celle qu'a la relation *meets* d'Allen. En effet en TempOQL $x_1$ meets $x_2 \equiv \mathsf{max}(x_1) + 1 = \mathsf{min}(x_2)$ alors que dans le système de relations d'Allen (adaptées au contexte d'un modèle de temps discret) $x_1$ meets $x_2 \equiv \mathsf{max}(x_1) = \mathsf{min}(x_2)$.

### 4.1.2 Manipulation d'historiques

TempOQL reprend les opérateurs du modèle d'historiques (Cf. § 3.1.3) en les "habillant" dans une syntaxe compatible avec le style d'OQL. En particulier les syntaxes des opérateurs d'ordre supérieur sur les historiques sont proches de celles des itérateurs OQL sur les collections (c.à.d. du select ... from ... where ... avec ou sans group by).

La définition formelle des constructions syntaxiques relatives aux historiques est détaillée dans l'annexe B.1. Ci-dessous nous illustrons les fonctionnalités offertes par certains de ces opérateurs au moyen de quelques requêtes simples portant sur la base de données considérée dans les chapitres précédents.

### Restrictions

Tout d'abord, nous illustrons la syntaxe de l'opérateur de restriction selon l'image $\Sigma_{im}$. Cette syntaxe s'inspire de celle des clauses from et where d'OQL, en particulier pour ce qui concerne l'utilisation du mot-clé as.

**Q6 : Restriction selon l'image**

*Quand l'employé de nom "X" gagne-t-il plus de 2000?*

```
/* type du résultat: EDI */
element(
select domain(e.salaire as s when s > 2000)  /* domain(H) dénote le domaine de l'historique H */
from LesEmployés as e
where e.nom = "X")
```

Suivant l'esprit d'OQL, la condition dans la clause when peut être une requête arbitrairement complexe. En particulier, il est possible d'utiliser d'autres opérateurs sur les historiques dans la description de cette condition, comme l'illustre la formulation suivante de la requête **Q5**. On pourra remarquer dans cette formulation que la syntaxe de l'opérateur permettant d'accéder à la valeur d'un historique à un instant donné, est la même que celle de l'opérateur OQL d'accès à un élément d'une liste[1].

**Q5** *(reprise)* **: Restriction selon l'image et accès à la valeur d'un historique à un instant.**

*Pour chaque employé, donner l'ensemble d'instants auxquels cet employé gagne plus de ce qu'il gagne le 1/2/99?*

```
/* type du résultat : bag<struct<employé: Employé, instants: EDI>> */
select employé: e,
       instants: domain(e.salaire as s when s > e.salaire[@"1/2/99"])
       /* H[I] dénote la valeur de l'historique H à l'instant I */
from LesEmployés as e
```

L'opérateur de restriction selon le domaine est noté during et possède une syntaxe "infixe". Son paramètre gauche est un historique et son paramètre droit est un ensemble d'instants. À nouveau, cet ensemble d'instants peut être obtenu au travers d'une requête faisant intervenir d'autres opérateurs sur les historiques, comme c'est le cas dans l'exemple suivant.

**Q7 : Composition des deux types d'opérateurs de restriction**

*Retrouver l'historique de la qualité de l'unité "X", lorsque sa production est supérieure à 3000.*

1. En OQL, l'expression L[i] extrait le (i-1)-ème élément d'une liste L.

```
/* type du résultat : Historique */
element(select u.qualité during domain(u.production as p when p > 3000))
        from LesUnités as u
        where u.codeUnité = "X")
```

### Transformation d'historiques et produits

L'opérateur H as x when P(x) présenté ci-dessus est une sorte de cas particulier d'une construction plus complexe de la forme map ... on ... when ... dont la sémantique combine une restriction avec une "transformation" d'un historique par application d'une fonction à chacune de ses valeurs. Plus précisément l'identité suivante lie ces deux opérateurs :

```
q1 as x when q2 ≡ map x on q1 as x when q2
```

Malgré la ressemblance entre la structure syntaxique du map/on/when et celle du filtre select/from/where il existe des différences importantes entre ces constructions. Premièrement le map/on/when construit un historique et non pas une collection comme le select/from/where. Deuxièmement la clause on ne fait intervenir qu'une seule requête de type historique alors que dans la clause from plusieurs requêtes de type collection peuvent apparaître avec une sémantique d'itérations emboîtées.

Pour exprimer des requêtes où plusieurs historiques sont comparés ou fusionnés il faut explicitement faire appel aux opérateurs de produit d'historiques (interne ou externe). La syntaxe de ces deux opérateurs est analogue à celle du constructeur de n-uplets struct d'OQL. Ce choix se justifie du fait que les valeurs de l'historique résultant d'un produit sont des n-uplets. La distinction entre le produit interne et externe se réflète syntaxiquement par l'utilisation des mots-clés join et ojoin respectivement. Plus précisément :

- join(a1: h1, a2: h2) dénote le produit interne des historiques h1 et h2. Si h1 est de type Historique<T1> et h2 de type Historique<T2> alors les valeurs de l'historique résultant sont de type *struct<a1 : T1, a2: T2>*.
- ojoin(a1: h1, a2: h2) dénote le produit externe de h1 et h2. Les mêmes règles de typage que pour le join s'appliquent.

### Q8 : Produit interne et transformation d'historiques.

*Pour chaque unité de production, donner l'historique de son volume de production pondérée par la qualité, restreinte aux instants où cette production pondérée est supérieure à 5000.*

```
/* type du résultat: struct<unité: UnitéDeProduction, production_pondérée: Historique<float>>*/
select struct(unité: u,
              production_pondérée: map c.prod * c.qual
                                   on join(prod: u.production, qual: u.qualite) as c
```

```
                when c.prod * c.qual > 5000
  from LesUnités as u
```

**Q9 : Produit interne.**

*Quand l'employé de nom "X" a-t-il gagné plus que l'employé de nom "Y".*

```
/* type du résultat: EDI */
element(select domain(join(sal_x: x.salaire, sal_y: y.salaire)
                      as c when c.sal_x > c.sal_y)
        from LesEmployés as x, LesEmployés as y
        where x.nom = "X" and y.nom = "Y")
```

Dans cette dernière expression, l'utilisation du produit externe au lieu du produit interne permet de prendre en compte le cas où X possède un salaire alors que Y n'en possède pas.

**Q10 : Produit externe.**

*Quand l'employé de nom "X" a-t-il gagné plus que l'employé de nom "Y". Aux instants où "X" (resp. "Y") possède un salaire alors que "Y" (resp. "X") n'en possède pas, on considère que le salaire de "Y" (resp. "X") est égal à la valeur neutre (en l'occurrence zéro).*

```
/* type du résultat: EDI */
element(select domain(ojoin(sal_x: x.salaire, sal_y: y.salaire)
                      as c when c.sal_x > c.sal_y)
        from LesEmployés as x, LesEmployés as y
        where x.nom = "X" and y.nom = "Y")
```

### Généralisation des opérateurs d'OQL aux historiques

Pour chaque opérateur d'OQL, TempOQL fournit un opérateur avec la même syntaxe mais s'appliquant aux historiques. La sémantique des opérateurs ainsi "généralisés" est définie en appliquant le principe suivant : étant donné un opérateur n-aire $\theta$: $\tau_1, \ldots \tau_n \rightarrow \tau_{n+1}$, on définit un opérateur $\theta$: History$<\tau_1>, \ldots$ History$<\tau_n> \rightarrow$ History$<\tau_{n+1}>$ tel que :

$$\forall i \in \text{Domaine}(h_1 \cap \ldots \cap h_n) \quad \text{Valeur}(h_1\ \theta\ \ldots\ \theta\ h_n, i) = \text{Valeur}(h_1, i)\ \theta\ \ldots\ \theta\ \text{Valeur}(h_n, i)$$

Ce principe de généralisation est à la base de certains langages à flots de données (p. ex. LUSTRE [CPHP87]). Il a aussi été appliqué sous diverses formes dans la conception de langages de requêtes temporels [SBJS98] et spatio-temporels [GBE+98].

Grâce au principe de généralisation, il est possible d'exprimer des requêtes mettant en correspondance plusieurs historiques, sans passer par l'opérateur join. De nombreuses requêtes se trouvent ainsi simplifiées, comme c'est le cas par exemple de la requête **Q9** que nous venons de formuler à l'aide du join.

**Q9** *(reprise)* : **Généralisation point-par-point.**
*Quand l'employé de nom "X" a-t-il gagné plus que l'employé de nom "Y".*

```
element(select domain(x.salaire > y.salaire as bool when bool)
        from LesEmployés as x, LesEmployés as y
        where x.nom = "X" and y.nom = "Y")
```

Dans cette formulation, x.salaire > y.salaire s'évalue en un historique de booléens. L'historique ainsi obtenu est alors restreint aux instants où il prend la valeur true.

Quelques opérateurs d'OQL échappent au principe de généralisation. Ceci est le cas notamment des constructeurs de valeurs de types paramétrés : struct, set, bag, list et array. Ces opérateurs ont la particularité d'être définis indépendamment du type de leur(s) paramètre(s). Ainsi, l'expression struct(l1: q1, l2: q2) est une requête bien typée en OQL, quelque soit le type de q1 et/ou de q2, y compris lorsqu'il s'agit d'une instanciation du générateur de types Historique<T>. Si le principe de généralisation était appliqué à l'opérateur struct, cette expression serait ambiguë : sans appliquer le principe de généralisation, le résultat serait un couple d'historiques ; en l'appliquant la requête s'évaluerait en un historique de couples. Plus précisément, en appliquant le principe de généralisation à l'opérateur struct on obtient un opérateur avec la même sémantique que celle du join.

Concrètement, considérons l'expression suivante :

```
element(select struct(production: u.production, qualité : u.qualité)
        from LesUnités as u
        where u.codeUnité = "U101")
```

En OQL non-étendu, le résultat de cette expression est un couple composé de l'historique de la production et de celui de la qualité de l'unité U101. Si nous appliquons le principe de généralisation à l'opérateur struct, cette expression s'évalue en un historique de couples : la valeur de cet historique à un instant donné (soit l), correspond à la juxtaposition des valeurs des historiques de la production et de la qualité de l'unité U101 à l'instant l (ce qui correspond exactement à la sémantique du join).

Si la seconde sémantique était adoptée, il ne serait plus possible de formuler une requête avec la première sémantique. Si par contre nous adoptons la première sémantique (celle d'OQL non-étendu), on peut toujours formuler une requête avec la seconde sémantique au moyen de l'opérateur join. Par conséquent, nous adoptons la première sémantique, ce qui revient à dire que l'opérateur struct n'est pas "généralisé" en TempOQL.

Une autre exception au principe de généralisation concerne les propriétés de type historique. Pour comprendre les raisons qui nous amènent à ne pas généraliser ces propriétés (vues comme des opérateurs), on considère l'expression suivante :

```
element(select o.superviseur.salaire
        from LesUnités as o
        where o.codeUnité = "U10")
```

Si le principe de généralisation était appliqué aux opérateurs superviseur et salaire, le résultat de cette expression serait de type Historique<Historique<Float>>. La valeur de cet historique à un instant donné (soit I), serait alors l'historique du salaire de l'employé qui supervise l'unité "U10" à l'instant I. En effet, l'expression o.superviseur s'évalue en un historique de superviseurs. En appliquant l'opérateur salaire à cet historique de superviseurs, on obtient bien un historique d'historiques de flottants, dont la valeur à un instant donné correspond à l'historique des salaires d'un superviseur.

L'expérience montre qu'un tel résultat est d'un intérêt très limité. A contrario, il serait intéressant que l'expression ci-dessus retrouve l'historique des salaires du superviseur de l'unité U10. En d'autres termes, on voudrait que l'opérateur salaire, appliqué à un historique de superviseurs HS, retrouve un historique de flottants, dont la valeur à un instant donné (soit I) soit égale au salaire du superviseur HS[I][2].

Pour obtenir un tel opérateur, nous définissons la généralisation aux historiques d'une propriété temporelle p (c.à.d. l'opérateur noté H.p) comme suit :

$$\forall\, i \in \text{Domaine}(H) \quad i \in \text{Domaine}(\text{Valeur}(H, i).p) \Rightarrow \text{Valeur}(H.p, i) = \text{Valeur}(\text{Valeur}(H, i).p, i)$$

On obtient ainsi un opérateur de *navigation point-par-point au travers d'historiques*, permettant d'exprimer de façon concise et à notre avis élégante, les requêtes formulées en OQL dans le § 2.2.3 pages 25 et 26. Ces requêtes ont aussi été formulées dans le langage de requêtes de TOOBIS, dans le § 2.3.3, pages 38 et 39.

**Q2** *(reprise)* : **Généralisation des propriétés temporelles. Application aux requêtes de navigation point-par-point.**

*Pour chaque ouvrier, donner l'historique de ses superviseurs.*

```
/* type du résultat : struct<ouvrier: Ouvrier, superviseur: Historique<Superviseur>> */
select struct(ouvrier: o, superviseur: o.unité.superviseur)
from LesOuvriers as o
```

**Q3** *(reprise)* : **Généralisation des propriétés temporelles. Application aux requêtes de navigation et de comparaison point-par-point.**

*À quels instants, l'unité supervisée par l'employé de nom X a-t-elle eu une production, pondérée par la qualité, supérieure à celle supervisée par l'employé de nom Y.*

```
/* type du résulat: EDI */
```

2. On rappelle que si H est un historique, H[I] dénote la valeur de cet historique à l'instant I.

```
element(select domain(superX.supervise.production * superX.supervise.qualite
                      > superY.supervise.production * superY.supervise.qualite
                      as bool when bool)
        from LesSuperviseurs as superX, LesSuperviseurs as superY
        where superX.nom = "X" and superY.nom = "Y")
```

Les remarques que nous venons de faire au sujet de la généralisation des propriétés de type historiqueΓs'appliquent aussi à la généralisation des méthodes dont le type du résultat est historique. Par exempleΓsi supervise et production étaient des méthodes et non pas des propriétésΓla sémantique de l'expression ci-dessus serait toujours la même.

### Regroupement

De même que le select ... from ... where ... défini par OQLΓla construction syntaxique map ... on ... when ... Γpossède deux clauses optionnelles group by et having. En la présence de la clause group byΓla sémantique du map ... on ... est définie à partir des opérateurs de regroupement sur les historiques (Cf § 3.1.3).

Plus précisémentΓlorsque l'expression apparaissant derrière le group by est de type GranularitéΓla sémantique de cette construction est définie à partir de l'opérateur Γ : l'historique issu de la clause onΓéventuellement restreint par la condition du whenΓest partitionné en un historique d'historiques ; chacun des sous-historiques générés pouvant être référencé dans les clauses having et map au moyen du mot-clé partition.

Définie de cette manièreΓla construction syntaxique map ... on ... group by ... permet d'exprimer des requêtes mettant en jeu des changements de granularité.

**Q11 : Changement de granularité**

*Pour chaque unité de production, et pour chaque mois où cette unité est en production, retrouver le nombre de jours dans ce mois où son volume de production est supérieur à 100.*

```
/* type du résultat: bag<struct<unité: UnitéDeProduction,production: Historique<Durée>>> */
select struct (unité: u
               production: map duration(partition)
                           /* partition est un historique à la granularité du jour;
                           duration appliqué à un historique retrouve la durée de son domaine. */
                           on u.production as prod
                           when prod > 100
                           group by Mois)
from LesUnités as u
```

Lorsque l'expression apparaissant derrière le group by est de type duréeΓla sémantique d'une requête de la forme map ... on ... group by ... est définie à partir de l'opérateur $\Delta$ (voir annexe B.1 pour plus de précisions).

**Q12 : Regroupement par intervalles de durée fixe.**

*Pour chaque unité, et pour chaque période de 10 jours consécutifs tel que le volume de production total de l'unité sur cette période est supérieur à 1000, donner la moyenne de ses volumes de production sur cette période.*

```
/* type du résultat: bag<struct<unité: UnitéDeProduction, prod_moy : Historique<float>>> */
select struct (unité: u,
               prod_moy : /* L'expression suivante construit un historique à la granularité du jour.
                             Pour chaque jour dans son domaine, la valeur de cet historique est la
                             production moyenne de l'unité u sur la période de 10 jours débutant ce
                             jour-là, à condition que la somme de ses productions sur cette période
                             soit supérieure ou égale à 1000 */
                          map havg(partition) /* havg calcule la moyenne d'un historique */
                          on u.production as prod
                          group by #"10 jours"
                          having hsum(partition) > 1000
from LesUnités as u
```

Les opérateurs havg et hsum utilisés dans cette expression retrouvent respectivement la moyenne et la somme des valeurs successives (avec répétitions)Γprises par un historique. Ces opérateurs sont définis à partir de leurs versions cumulatives cavg et csum (Cf. annexe B.1).

### Découpage d'historiques

Les opérateurs AprèsPremierΓAvantPremierΓAprèsDernier et AvantDernier du modèle d'historiquesΓse retrouvent en OQL sous forme d'opérateurs nommés respectivement afterfirst beforefirstΓafterlast et beforelastΓtous munis d'une syntaxe calquée sur celle du when.

Ces opérateurs permettent de raisonner sur la succession des valeurs d'un historique. Par exempleΓdans la requête ci-aprèsΓl'historique du superviseur associé à chaque ouvrier est restreint aux instants suivant la première fois où une certaine condition est vérifiée.

**Q13 : Découpage d'historiques. Application au raisonnement sur la succession.**

*Quelles unités ont eu comme superviseur "X" et quelque temps après "Y".*

```
/* type du résultat: bag< UnitéDeProduction> */
select u
from LesUnités as u
where exists s2 in range(u.superviseur as s1 afterfirst s1.nom = "X") : s2.nom = "Y"
```

### Représentations par instants et par intervalles

Les opérateurs que nous venons de présenterΓpermettent de raisonner sur la succession dans le temps (des événements qui se succèdent) mais ils ne permettent pas d'exprimer les

notions de continuité et de changement de valeur. En particulier, il n'est pas possible au travers de ces opérateurs, d'exprimer des requêtes telles que *"combien de fois le salaire de l'employé de nom X a-t-il augmenté"*. ou alors *"quel est le plus long intervalle de temps au cours duquel le salaire de l'employé X est resté constant"*.

C'est justement pour exprimer ces notions qu'il est utile de raisonner sur une représentation à base d'intervalles "maximaux". TEMPOQL permet d'exprimer ce type de raisonnements au travers de l'opérateur xchronicle, dont la sémantique a été présentée dans le § 3.1.2. Étant donnée une requête H de type Historique<T>, l'expression xchronicle(H) est de type struct<moment : Interval, valeur : T>.

**Q14 : Représentation par intervalles. Application au raisonnement sur la continuité.**

*Qui a supervisé l'unité U de façon ininterrompue, plus longtemps que ne l'a fait l'employé X?*

```
/* type du résultat: set<Superviseur> */
select distinct instantané_sup.valeur
from UnitéDeProduction as u, xchronicle(u.superviseur) as instantané_sup
where u.codeUnité = "U"
   and duration(instantané_sup.moment) > max(select duration(instantané_X.moment)
                                             from xchronicle(u.superviseur) as instantané_X
                                             where instantané_X.valeur.nom = "X")
```

La représentation par instants de son côté, permet d'exprimer les requêtes d'agrégation tranversale point-par-point (Cf. § 2.3.1).

**Q4** *(reprise)* **: Représentation par instants. Application à l'agrégation transversale point-par-point.**

*Pour chaque journée où au moins une unité est en production, retrouver la somme des productions de toutes les unités ce jour-là.*

```
/* type du résultat: bag<struct<moment: Instant, valeur: short> */
select struct(moment: ip.moment, valeur: sum(select c.ip.valeur from partition as c))
from LesUnités as u, ichronicle(u.production) as ip
group by ip.moment /* ip.moment est de type Instant */
```

### 4.1.3 Description de motifs d'historiques

Par composition des quatre opérateurs de découpage d'historiques, il est possible d'exprimer des requêtes faisant intervenir des séquences d'événements telles que *"A puis B suivi de C"*, où A, B et C sont des conditions portant sur les valeurs d'un historique. Toutefois, les formulations ainsi obtenues deviennent vite illisibles lorsque des séquences complexes entrent

en jeu. Par ailleurs, si l'on rajoute des contraintes de durée entre des événements (p. ex. *"A puis B suivi de C dans un délai de 3 jours"*), il devient nécessaire de composer les opérateurs de découpage avec celui de regroupement suivant une durée, ce qui accroît encore la complexité des expressions.

Notre approche pour maîtriser cette complexité consiste à fournir un langage conçu spécifiquement pour la description de séquences d'événements avec contraintes de durées. Ce langage combine les opérateurs d'OQL avec des opérateurs issus des langages réguliers (concaténation et répétition). Les opérateurs d'OQL sont utilisés pour exprimer des conditions sur la valeur prise par un historique à un instant donné, alors que les opérateurs réguliers permettent de composer ces conditions pour exprimer différentes formes de séquencement.

Une expression de ce langage (appelée par la suite un *motif*) dénote un ensemble éventuellement infini d'historiques. Étant donnés un historique et un motif, une opération naturelle consiste à rechercher un sous-historique de l'historique en question, appartenant au motif. C'est d'ailleurs cette idée qui est à la base de nombreux logiciels de recherche sur des textes, comme par exemple les commandes grep et sed sous Unix.

Suivant ce principe, le type Historique en TEMPOS comporte un opérateur à valeur booléenne, prenant en argument un historique et un prédicat défini à partir d'un motif. Cet opérateur est repris en TEMPOQL sous forme d'un opérateur noté matches, muni d'une syntaxe analogue à celle du when.

La syntaxe et sémantique du langage de description de motifs et de l'opérateur matches sont spécifiées formellement dans l'annexe B.2. Ci-après, nous illustrons certains aspects de ce langage au travers d'exemples portant sur l'historique des superviseurs d'une unité u. Pour chaque énoncé en langue naturelle, nous donnons une expression TEMPOQL qui s'évalue à vrai si l'énoncé est vérifié.

- **Séquencement simple.**
  *L'employé "X" a remplacé "Y" au poste de superviseur de l'unité u.*

  ```
  u.superviseur as sup matches sup.nom = "X" followed by sup.nom = "Y"
  ```

- **Répétition sans contrainte de durée.**
  *L'unité u a eu comme superviseur "X" et quelque temps après "Y".*

  ```
  u.superviseur as sup matches sup.nom = "X" followed by * followed by sup.nom = "Y"
  ```

  Dans cette expresion, le symbole * est un raccourci du motif several true (Cf annexe B.2).

- **Répétition avec contrainte de durée maximale.**
  *L'employé "Y" a été nommé superviseur de l'unité u moins de trois jours après que "X" ait quitté ce poste.*

  ```
  u.superviseur as sup matches sup.nom = "X"
  ```

followed by true during < #"3 jours"
followed by sup.nom = "Y"

- **Répétition avec contrainte de durée minimale.**
  *L'employé "Y" a été nommé superviseur de l'unité u plus de trois jours après que "X" ait quitté ce poste.*

u.superviseur as sup matches sup.nom = "X"
followed by (sup.nom != "X" and sup.nom != "Y') during > #"3 jours"
followed by sup.nom = "Y"

Quelques langages similaires à celui que nous venons de présenter ont été étudiés dans le cadre des bases de données temporelles et vidéo. AinsiΓ[CG94] propose un langage de description de motifs spatio-temporels. Dans ce langageΓles motifs temporels sont décrits à partir de conditions "élémentaires" sur les valeurs de l'historique manipulé. Ces conditions élémentaires s'évaluent par rapport à un instant de référenceΓqui au départ est fixé comme étant le premier instant du domaine de l'historique. L'instant de référence est ensuite déplacé au travers d'opérateurs tels que "déplacement de 10 unités vers la droite"Γou "déplacement jusqu'au prochain instant où l'historique vérifie une condition donnée". La répétition s'exprime au travers de structures de contrôle proches de celles des langages impératifs : boucles de type tant que combinées à des expressions conditionnelles de type si ... alors ... sinon.

Nous pensons que la différence entre ce dernier langage est celui que nous proposonsΓ est comparable à celle qui existe entre un générateur d'analyseurs lexicaux tels que LexΓ et un langage de programmation impérative tel que C : il est bien sûr possible d'écrire un programme en C qui recherche un certain motif dans un texteΓmais on préfère souvent passer par Lex.

Dans le cadre de l'interrogation d'annotations de vidéosΓ[NYK97] propose un langage de recherche de motifs à base d'expressions régulières avec contraintes de durées. À la différence du langage que nous proposonsΓles termes de base de ce langage sont des événements représentés par un ensemble fini de constantesΓet non pas des conditions sur la valeur des annotations de la vidéo. Par ailleursΓpour des raisons d'implantation (celle-ci est fondée sur une classe d'automates proposée par les mêmes auteurs)Γle pouvoir d'expression du langage est très limité. En particulierΓil n'est pas possible d'exprimer des contraintes de durée minimale.

D'autres langages de description de suites d'événements à base d'expressions régulières ont été étudiés dans divers contextesΓcomme par exemple dans celui des Bases de Données Actives [Pat98].

## 4.2 Visualisation de bases d'objets temporels

Dans cette sectionΓnous nous intéressons aux techniques de visualisation interactive d'objets dites *généralistes*Γau sens où elles ne font pas d'hypothèses particulières sur la structure des objets manipulés. Nous étudions ensuite comment ces techniques se comportent lorsqu'elles opèrent sur des objets temporelsΓc'est-à-dire des objets munis de propriétés de type historique. EnfinΓnous proposons une technique de visualisation d'objets temporels inspirée d'un paradigme de navigation sur des collections d'objetsΓappelé "navigation synchrone". Cette section reprend partiellement le contenu de [DDFN00].

### 4.2.1 Visualisation de collections d'objets

Concernant les outils de visualisation de donnéesΓdes différences importantes séparent les SGBD relationnels de ceux à objets.

Dans les premiersΓles techniques de visualisation se limitent le plus souvent à la présentation des relations sous forme tabulaireΓdans la lignée du paradigme rendu populaire par les feuilles de calcul telles que LOTUS 1-2-3[3]. Les possibilités d'interaction sont généralement limitées à celles liées aux mises à jour. À cet égardΓseuls certains SGBD commerciaux tels qu'ACCESS[4] font exceptionΓen offrant la possibilité de visualiser "un par un" les n-uplets d'une relation dans des formulairesΓet de naviguer au travers des liens entre n-uplets codés par les contraintes de clé étrangère.

Dans les SGBD à objets en revancheΓl'unité de visualisation est avant tout l'objetΓreprésenté par un formulaire contenant des couples ⟨propriétéΓvaleur⟩Γen respectant les contraintes de visibilité des propriétés. Les associations entre les objets sont généralement rendues explicites par des arcsΓet les possibilités d'interaction incluent notamment la navigation et l'appel de méthodes (voir par exemple $O_2$Look [PBL+92]).

Remarquons au passage que ces différences se retrouvent aussi dans les langages de requêtes visuels. En effetΓdans les modèles relationnels ces langages reposent sur des représentations tabulaires (c'est le cas par exemple de QBE [Zlo75]). A contrarioΓles langages de requêtes visuels pour bases de données à objets s'appuient sur des formulaires dénotant des classesΓet exploitent le concept de navigation pour permettre l'expression de requêtes associatives (Cf. par exemple ODEVIEW [DGJS95]ΓSUPER [DAA+95] et VOODOO [Feg99]).

#### Navigation synchrone au travers de collections

La plupart des techniques de visualisation pour bases de données à objets s'inspirent de celles développées auparavant dans le cadre de modèles entité-association. Ceci est le cas

3. http://www.lotus.com
4. http://www.microsoft.com/office/access

notamment du concept de *navigation synchrone*[5] sur des collections d'objets, qui apparaît en particulier dans ODEVIEW [DGJS95], SUPER [DAA+95] et PESTO [CHMW96].

Le but principal de la navigation synchrone telle que définie dans PESTO[6] est de visualiser un ensemble de chemins de navigation partant d'une certaine collection d'objets. Un *chemin de navigation partant d'une classe* $C_1$ est défini comme une suite d'attributs $A_1.A_2...A_n$, tels que $A_1$ est un attribut de la classe $C_1$ dont le type est une classe $C_2$, $A_2$ est un attribut de $C_2$ dont le type est une classe $C_3$, etc. Dans la base de données modélisée par le schéma UML de la figure 1(a) (page 21), unité et unité.superviseur.salaire sont des exemples de chemins de navigation partant de la classe Ouvrier.

Une collection d'objets est représentée en PESTO sous forme d'un formulaire contenant un objet quelconque de la collection (on parlera par la suite de l'*objet de référence*), ainsi que des curseurs permettant le cas échéant d'avancer (revenir) à l'objet suivant (précédent) dans la collection. Lorsque la valeur d'un des attributs représentés dans un formulaire est de type objet, il est possible d'*activer* le bouton correspondant au travers d'un clique de souris. Le contenu de l'objet référencé est alors affiché dans une fenêtre séparée, reliée par un arc à la fenêtre à partir de laquelle l'activation est effectuée. De cette façon, un arbre de fenêtres partant de l'objet de référence est constitué (Cf. figure 1(a)). Lorsque l'utilisateur change l'objet de référence, le contenu de chacune des fenêtres dans cet arbre est éventuellement mis à jour en conséquence (Cf. figure 1(b)).

PESTO traite aussi le cas où l'un des attributs d'un chemin de navigation est de type "collection" (p. ex. l'attribut ouvriers de la classe Employé). Dans ce cas, le mécanisme décrit ci-dessus est appliqué récursivement : un objet de référence est attaché au formulaire permettant de visualiser le contenu d'un attribut de type collection.

La navigation synchrone dans le système SUPER est définie de façon similaire (même si les auteurs n'utilisent pas ce terme). Il est à noter par ailleurs que SUPER fournit deux modes de visualisation : un à base de formulaires (comme dans PESTO), et un autre à base de diagrammes de composition de type "entité-association".

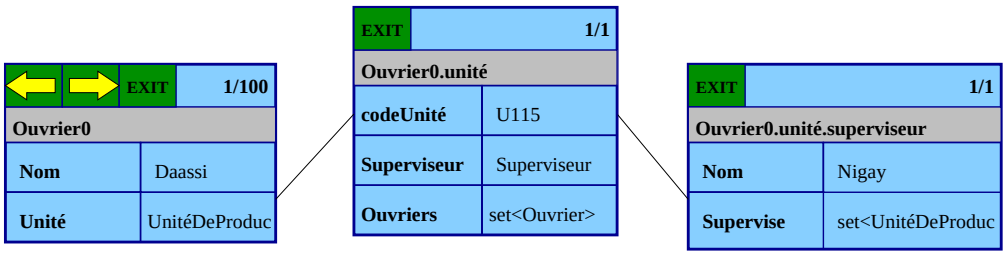
### Navigation au travers de collections en $O_2$Look

Dans le module de visualisation et d'édition d'objets $O_2$Look du SGBD $O_2$, une collection est représentée par un formulaire composé d'une liste de boutons, chacun dénotant un élément de la collection. Lorsque le type de ces éléments est une classe[7], chacun de ces boutons dénote

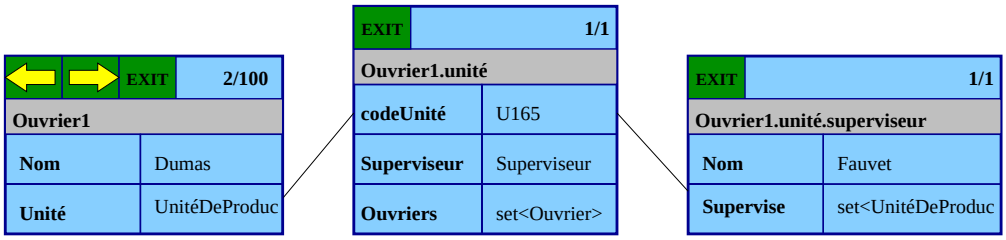
5. Le choix de ce terme est assez critiquable. En effet, l'adjectif "synchrone" qualifie ici les interactions entre les fenêtres composant une interface, alors que dans le domaine de l'interaction homme-machine, cet adjectif est utilisé pour qualifier les interactions entre des utilisateurs dans un environnement de type collecticiel.

6. Le concept de navigation synchrone défini en ODEVIEW est très similaire à celui-ci

7. Dans le modèle de données du SGBD $O_2$, une distinction forte est faite entre *type* et *classe*. Les instances de types sont appelées des *valeurs*, et celles de classes, des *objets*.




(a) Visualisation du premier employé de la collection




(b) Visualisation du deuxième employé de la collection

FIG. 4.1 – *Visualisation PESTO d'un chemin de navigation partant d'une collection d'employés. Le schéma des objets est spécifié dans la figure 1(b) (page 21).*

un objet sur lequel il est possible d'invoquer n'importe quelle méthode, et en particulier des méthodes d'affichage (méthode display) et d'édition (méthode edit). Par défaut, ces deux méthodes affichent un formulaire contenant des couples ⟨ propriété, valeur ⟩, en respectant les contraintes de visibilité des propriétés. Lorsque la valeur associée à une propriété est elle-même un objet, il est à nouveau possible d'invoquer une méthode de visualisation sur celui-ci. De cette manière, les méthodes display et edit du module $O_2$Look fournissent un mécanisme de navigation simple.

À la différence de PESTO, les fenêtres générées par $O_2$Look au cours d'une session sont indépendantes, au sens où une interaction portant sur une fenêtre donnée n'affecte pas les autres fenêtres. Toutefois, lorsqu'une fenêtre F est ouverte depuis une autre fenêtre F', le système impose que F soit fermée avant F' : les fenêtres sont donc implicitement organisées en une hiérarchie.

### 4.2.2 Navigation au travers de chroniques

La spécificité de la dimension temporelle rend la plupart des techniques de visualisation d'objets inadéquates lorsque ceux-ci comportent des historiques. Dans les paragraphes qui suivent, nous montrons en particulier les limitations de PESTO et d'$O_2$Look au regard de l'exploration d'historiques, représentés sous forme de chroniques : collections de couples

⟨instantΓobjet⟩ ou ⟨intervalleΓobjet⟩ (Cf. § 3.1.2).

### Le cas de PESTO

Le principe de navigation synchrone au travers de collectionsΓlorsqu'il est appliqué à la visualisation d'historiques représentés sous forme de chroniquesΓne reflète pas la notion de simultanéitéΓc'est-à-dire qu'il ne permet pas de naviguer aisément au travers de plusieurs historiques de façon "synchronisée". Par exempleΓdans la figure 4.2Γnous montrons un état possible d'une session en PESTO où l'on visualise des attributs de type chronique. Qu'elles soient liées ou pas entre elles ou pasΓles fenêtres contenant des chroniques peuvent faire référence à des instantanés ne partageant pas d'instants communsΓrendant ainsi malcommode la tâche consistant à observer simultanément l'évolution de ces deux propriétés. Plus précisémentΓon remarque que les estampilles temporelles des instantanés affichés dans les fenêtres 2Γ3 et 5Γne s'intersectent pas. Si l'utilisateur veut par exemple observer l'état des objets à une date donnéeΓdisons le 1/1/90Γil doit interagir avec les boutons libellés par des flèches des fenêtres 3 et 5.

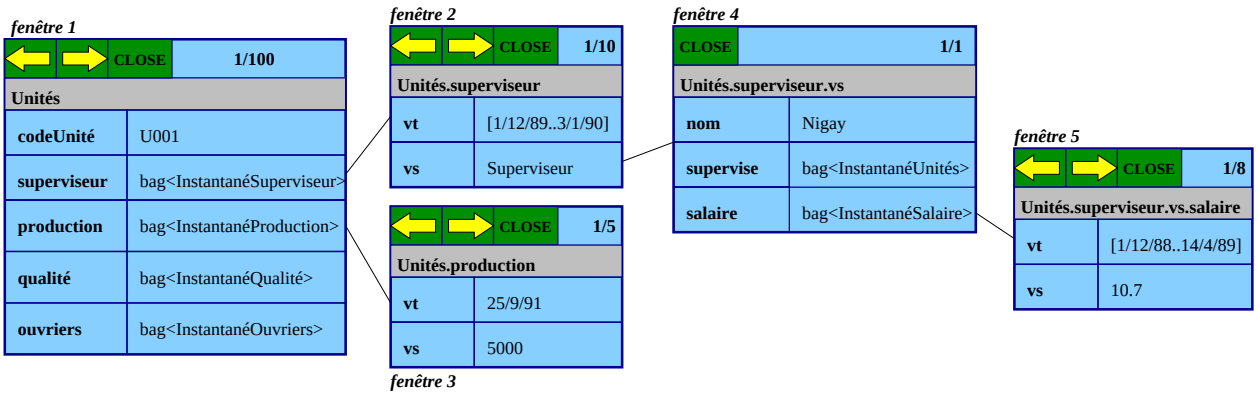



FIG. 4.2 – *Application de la navigation synchrone aux chroniques. Le schéma des objets est celui décrit dans la figure 2(b) page 23. En particulier, les historiques sont représentés par des collections d'instantanés.*

### L'approche $O_2$Look

En $O_2$LookΓles méthodes d'affichage et d'édition peuvent être programmées de façon à fournir des représentations visuelles adaptées à des types de données autres que ceux fournis par le SGBD. Ce mécanisme d'extensibilitéΓcouplé au fait qu'il est possible d'inclure n'importe quel "widget" Motif[8] au sein d'une présentation $O_2$LookΓrend envisageable l'intégration dans ce système de n'importe quelle technique de visualisation d'une chronique

8. http://www.motifzone.com

donnée. À titre d'exemple, la figure 4.3 illustre l'intégration d'une technique de visualisation de chroniques. L'appel à la méthode display depuis la fenêtre présentant un ouvrier donne naissance à la fenêtre présentant un historique d'unités de production. Sur cette fenêtre, l'utilisateur peut à nouveau appeler la méthode display donnant naissance à la fenêtre présentant une unité de production.

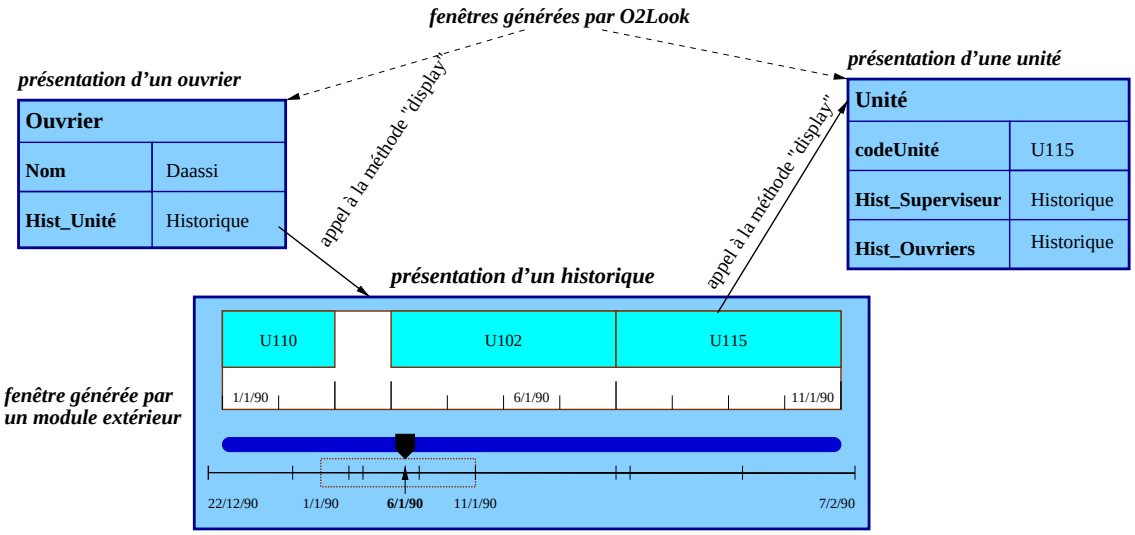



FIG. 4.3 – *Intégration d'une technique de visualisation de chroniques dans $O_2$Look. Le schéma des objets est celui décrit dans la figure 3.9 page 53. En particulier, les historiques sont des instances du type* Historique *du modèle* TEMPOS.

Ces mécanismes d'extensibilité ne suffisent pourtant pas à intégrer dans $O_2$Look des techniques de visualisation qui traitent la dimension temporelle de façon orthogonale aux autres dimensions des objets. En particulier, pour contrôler la manière dont les objets comportant des propriétés temporelles sont présentées (par exemple pour afficher leur valeur à un instant donné au lieu de tout leur historique), il est nécessaire de programmer entièrement les interfaces.

### 4.2.3 Visualisation point-par-point d'objets temporels.

Il ressort des paragraphes précédents que dans la plupart des techniques de visualisation pour bases d'objets, l'interaction avec l'utilisateur est fondée sur la *navigation* : navigation au travers des objets composant une collection, ou navigation d'un objet à un autre au travers de ses propriétés.

Lorsqu'on rajoute une dimension temporelle, il devient envisageable de naviguer aussi au travers du temps. Ainsi, dans la figure 4.3 nous avons schématisé une façon de "naviguer" au travers de l'historique d'une instance de propriété temporelle. En d'autres termes, en fixant un objet et l'une de ses propriétés, il est possible de naviguer au travers du temps, pour

connaître les valeurs prises cette propriété. Réciproquement, il serait envisageable en fixant un instant dans le temps, de naviguer au travers des états d'une collection d'objets et de leurs propriétés à cet instant. Ceci permettrait à l'utilisateur d'avoir une image de l'état d'une portion de la base de données à un instant, et éventuellement d'en observer l'évolution au cours du temps.

Sur la base de cette observation, nous développons dans la suite un paradigme de visualisation interactive d'objets temporels. Ce paradigme reprend les principales idées de la "navigation synchrone" sur des collections présentée ci-haut, en mettant l'accent sur l'orthogonalité des dimensions structurelles et temporelles des objets. Conformément à la remarque de la note de bas de page No. 5, page 88, nous préférons l'adjectif "point-par-point" à celui de "synchrone", pour qualifier cette technique.

### Aperçu général

L'interface de la technique de visualisation point-par-point est constituée d'une fenêtre comportant une droite temporelle, et d'un arbre de fenêtres dénotant des états d'objets temporels. Chacune des fenêtres dans cet arbre est appelée une *fenêtre-état*. L'objet associé à la fenêtre-état racine de l'arbre, est appelé l'*objet principal* (Cf. figure 4.4).

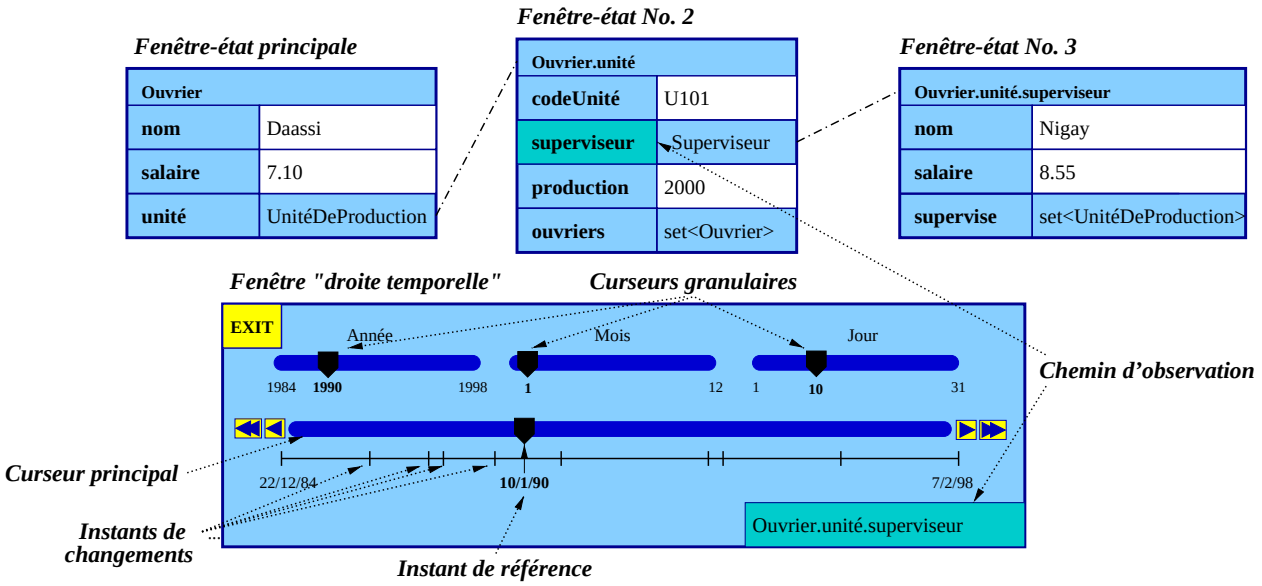



FIG. 4.4 – *Exemple de visualisation point-par-point. Le schéma de la base de données sous-jacente peut être soit celui de la figure 3.9 page 53, soit celui spécifié dans le paragraphe 3.2.4. En d'autres mots, les propriétés dont l'évolution est observée, sont soit des propriétés de type* Historique, *soit des propriétés temporelles.*

Au début d'une session, l'arbre de fenêtres-états comporte un seul noeud, correspondant à l'objet principal. L'utilisateur peut ensuite ajouter des nouvelles fenêtres-états en interagis-

sant avec celles déjà sur place. Chaque fenêtre-état reflète l'état à un instant dit *de référence* d'un chemin de navigation partant de l'objet principal et dont certains des composants peuvent être des propriétés temporelles[9].

La fenêtre "droite temporelle" comporte une barre de défilement sur laquelle se balade un *curseur* dit *principal*. Ce curseur fixe l'instant de référence. Au début d'une session le curseur principal se trouve au milieu de sa barre de défilement. Sa position varie ensuite suivant les interactions de l'utilisateur avec les éléments de la fenêtre "droite temporelle". Parmi ces éléments on retrouve en particulier une collection de barres de défilement placées au dessus de la barre de défilement principale sur lesquelles se déplacent des curseurs dits "granulaires". Ces curseurs permettent de repérer l'instant de référence par rapport à des conventions calendaires fixées par l'utilisateur. Par exemple dans l'exemple de la figure 4.4 l'instant de référence est repéré par rapport au système de granularités [Année Mois Jour].

Les boutons situés aux extrémités du curseur principal constituent un autre dispositif de déplacement de l'instant de référence. Les deux boutons étiquetés par des flèches simples permettent de déplacer l'instant de référence d'une position vers la gauche ou vers la droite. Ceux étiquetés par des doubles flèches déplacent le curseur principal jusqu'au prochain (resp. précédent) instant où un changement a lieu sur la valeur de l'un des chemins de navigation visualisés (appelé le *chemin principal d'observation*). Les *instants de changement* de la valeur de ce chemin sont affichés sous forme de marques verticales traversant une ligne horizontale placée au dessous du curseur principal.

Dans la figure 4.4 le chemin principal est ouvrier.unité.superviseur. Entre deux instants de changement le superviseur de l'ouvrier de nom "Daassi" reste le même.

On remarque que les fenêtres-états sont dépendantes de la fenêtre "droite temporelle" : toute modification de l'instant de référence étant susceptible d'entraîner un changement de l'état qu'elles dénotent.

### Structure et comportement de la fenêtre "droite temporelle"

La fenêtre "droite temporelle" est essentiellement composée d'un curseur dit *principal* que l'utilisateur peut déplacer le long d'un intervalle appelé le *domaine de navigation temporelle*. Le domaine de navigation temporelle est soit la couverture[10] du domaine temporel de l'objet principal (s'il s'agit d'une instance d'une classe temporelle) soit la couverture de l'union des domaines temporels des propriétés de cet objet (dans tous les autres cas). Par exemple si l'objet principal est un ouvrier O tel que:

O.salaire = set( struct(moment: [1..4], valeur: 10.0), struct(moment: [6..9], valeur : 12.0) ) et
O.unité = set( struct(moment: [2..4], valeur: X), struct(moment: [6..8], valeur : Y) )

9. Ici, les propriétés de type historique sont assimilées aux propriétés temporelles.
10. La couverture d'un ensemble d'instants est le plus petit intervalle le contenant.

(où X and Y dénotent des unités de production)Γalors le domaine de navigation est l'intervalle [1..9].

Le concept de *système de repérage multigranulaire* facilite les déplacements au travers du domaine de navigation temporelleΓen permettant de repérer un instant par les composants de sa forme multigranulaire dans un certain système de granularités. Nous donnons des définitions concises de ces deux derniers conceptsΓen renvoyant le lecteur à [Can97] pour plus de détails.

**Definition 6** (*Système de granularités*). Un *système de granularités* est une liste non-vide de granularités comparables triées en ordre décroissant suivant la relation $\prec$. □

Par exempleΓla liste [Année, Mois, Jour] est un système de granularités.

**Definition 7** (*Forme multigranulaire*). La *forme multigranulaire* FM d'un instant I dans un système de granularités SG est définie récursivement comme suit :

- Si $SG = [G_1]$ alors $FM = [E_1 + 1]$ où $E_1$ est la position de I.
- Si $SG = [G_1, G_2, \ldots G_n]$ alors $FM = [E_1 + 1, E_2 + 1, \ldots E_n + 1]$ où $[E_1, E_2, \ldots E_{n\text{-}1}]$ est la forme multigranulaire de l'approximation de I dans $G_{n\text{-}1}$ (soit I' cette approximation) et $E_n$ est la distance entre I et la borne inférieure de l'expansion de g' dans $G_n$. C'est-à-dire :

  $E_n$ = I - min(expand(approx(I, $G_{n\text{-}1}$),$G_n$))

  On rappelle que les opérateurs min sur les EDIΓde même qu'approx et expand sur les instantsΓont été introduits dans les § 3.1.1 et 4.1.1

□

Par exempleΓ[1999, 8, 3] dénote la forme multigranulaire de l'instant @"3/8/1999" dans le système de granularités [Année, Mois, Jour].

Un système de repérage multigranulaire dont le système de granularités est SGΓcomporte autant de barres de défilement qu'il y a de granularités dans SG. La position du curseur sur chacune de ces barres fixe la valeur d'une des composantes de la forme multigranulaire de l'instant de référence. AinsiΓil est possible d'avancer et de reculer l'instant de référence avec des "pas" plus ou moins grands selon la composante de la forme multigranulaire sur laquelle on agit.

EnfinΓ lorsqu'un chemin principal d'observation a été sélectionnéΓil est aussi possible de modifier l'état du curseur principal au travers de deux flèches situées aux extrémités de celui-ci (voir figure 4.4) : la double flèche droite (resp. gauche) ayant pour effet de déplacer le curseur au prochain (resp. précédent) instant de changement.

Les diagrammes UML fournis dans l'annexe C.1Γdécrivent plus précisément la structure de la fenêtre droite-temporelle et les interactions entre ses composants.

### Structure et comportement des fenêtres-états

Une fenêtre-état est un formulaire dénotant l'état d'un objet à l'instant de référence. La notion d'état d'un objet est définie à partir des états de ses propriétés comme suit.

**Definition 8** (*État d'un objet et de ses propriétés à un instant*). L'état d'un objet à un instant donné est la composition des états de chacune de ses propriétés définies à cet instant. Une propriété fugitive de type autre qu'historique est *définie* à tout instant et son état est donné par sa valeur. Étant donné un objet O une propriété temporelle ou une propriété fugitive de type historique portant sur cet objet est *définie* à un instant I si et seulement si le domaine temporel de O.P contient I. L'état de cette propriété à l'instant I est alors donné par l'expression Valeur(O.P, I) (en supposant que le mode d'accès est "temporel") □

Dans une fenêtre-état chaque ligne correspond à une propriété de l'objet visualisé. Une ligne est constituée de deux boutons disposés côte à côte. Le bouton gauche est libellé par le nom de la propriété. Lorsque l'utilisateur clique sur un tel bouton le chemin qui part de l'objet principal et qui aboutit sur la propriété en question devient le chemin d'observation par rapport auquel les instants de changement sont déterminés. Par exemple le fait de cliquer sur le bouton gauche de la deuxième ligne de fenêtre-état No. 3 figure 4.4 fixe le chemin d'observation à Ouvrier.unité.superviseur.salaire.

Le bouton gauche dans une ligne dénote la valeur de la propriété dénotée. Lorsqu'une propriété est de type littéral sa valeur est affichée directement sur le bouton correspondant (boutons colorés en blanc dans la figure 4.4). Ce bouton n'est alors pas "cliquable". Par contre lorsque la valeur de la propriété est un objet ou une collection le bouton est libellé par le type de cette valeur et le bouton est "cliquable". Lorsqu'un tel bouton est cliqué un nouveau noeud est ajouté à l'arbre de fenêtres-états dont le père est la fenêtre-état à partir de laquelle l'interaction est effectuée. Cette nouvelle fenêtre-état contient l'état à l'instant de référence de l'objet accessible à partir de l'objet père au travers de la propriété activée. La définition ci-après précise ce qu'on entend par "accessible".

**Definition 9** (*Accessibilité entre objets au travers d'une propriété temporelle*). Un objet O' est dit *accessible* à partir d'un objet O au travers d'une propriété temporelle (ou de type historique) P à un instant I si VS(O.P, I) = O' □

Au fur et à mesure que l'utilisateur navigue d'un objet à un autre au cours d'une session un arbre de fenêtres-états se constitue. Chacune de ces fenêtres-états correspond alors à un chemin de navigation partant de l'objet principal. Plus précisément une fenêtre-état dont le chemin de navigation est P1...Pn représente l'état à l'instant de référence de l'objet accessible à cet instant depuis l'objet principal au travers du chemin P1...Pn. La notion d'accessibilité entre objets à partir d'un chemin de navigation est définie par généralisation de la définition

précédente.

**Definition 10** (*Accessibilité entre objets au travers d'un chemin de navigation temporel à l'instant I*). Un objet O' est dit *accessible* à partir de l'objet O au travers du chemin de navigation P1.P2...Pn à l'instant IΓsi et seulement si O' est accessible depuis l'objet VS(O.P1, I) au travers du chemin de navigation P2...Pn à l'instant I. Si I n'appartient pas au domaine temporel de O.P1Γalors le chemin P1.P2...Pn partant de l'objet O est dit *indéfini*. □

Après chaque interaction avec la fenêtre "droite temporelle"Γle nouvel instant de référence est communiqué à la fenêtre-état principale. Celle-ci actualise alors l'état de l'objet qu'elle affiche. Au cours de cette actualisationΓsi la valeur de l'une de ses propriétés change et que cette propriété est activéeΓla nouvelle valeur est communiquée à la fenêtre-état fille attachée à cette propriété. EnsuiteΓla fenêtre-état principale notifie le nouvel instant de changement à toutes ses fenêtres-fillesΓet le processus ci-dessus est effectué recursivement sur chacune d'elles. Ceci est illustré dans la figure 4.5

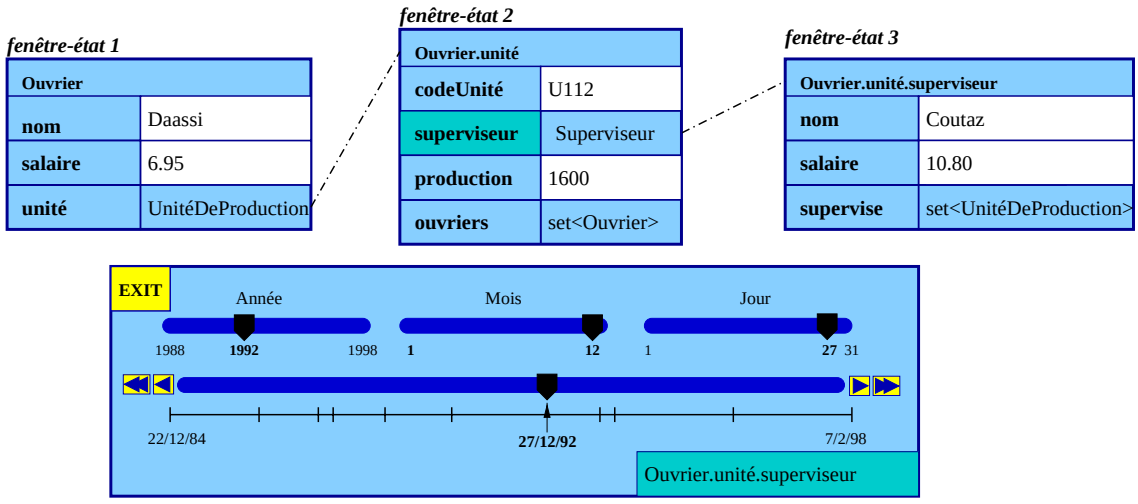



FIG. 4.5 – *Modification de l'instant de référence sur la configuration de la figure 4.4.*

Lorsqu'un chemin de navigation présent dans une visualisation point-par-point devient indéfini suite à un déplacement de l'instant de référenceΓla fenêtre-état associée à ce chemin est momentanément désactivée. L'effet de la désactivation d'une fenêtre dépend des choix d'implantation et/ou de la configuration utilisateur. Nous distinguons deux alternatives :

- Faire disparaître de l'écran les fenêtres désactivées.
- Introduire un trait visuel distinctif qui indique la désactivation d'une fenêtre-état. Par exempleΓla figure 4.6 montre une approche consistant à griser les noms des attributs et à supprimer les labels dénotant leurs valeurs.

Nous pensons que la deuxième alternative est préférableΓcar la disparition/réapparition

de fenêtres au cours d'une session tend à violer le principe ergonomique dit de "continuité visuelle" [Cou90].

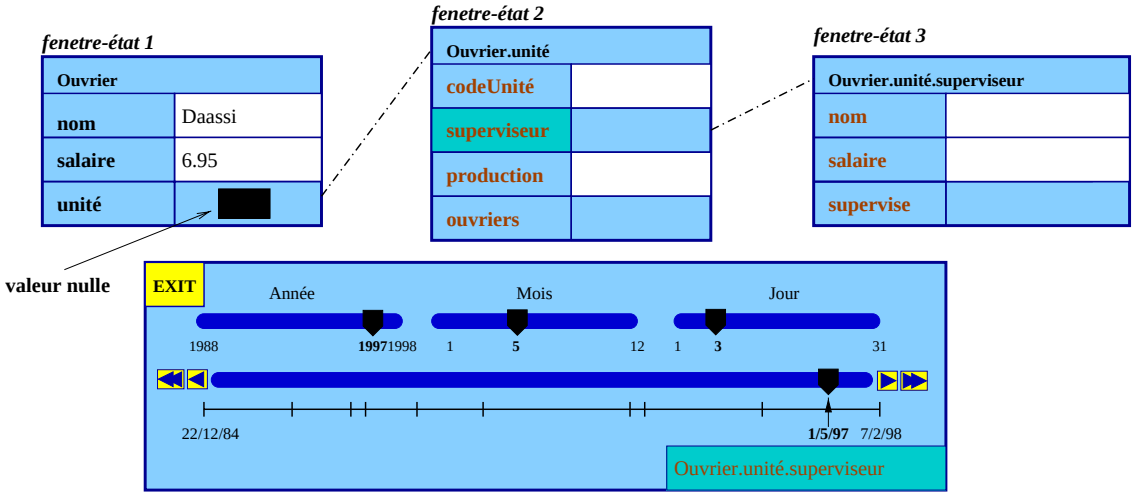



FIG. 4.6 – *Exemple de visualisation point-par-point. Les fenêtres 2 et 3 sont désactivées.*

On peut remarquer que la fenêtre-état principale peut aussi être désactivéeΓpuisque le domaine de navigation temporelle peut contenir des instants n'appartenant pas au domaine d'observation de l'objet principal. Par exempleΓsupposons que le domaine d'observation de l'objet principal est { [1..3], [5..7] }. Le domaine de navigation temporelle est alors la couverture de cet EDIΓsoit [1..7]. Il s'en suit que la fenêtre-état racine devient inactive si l'utilisateur fixe l'instant de référence à 4.

Les diagrammes UML fournis dans l'annexe C.2Γdécrivent plus précisément la structure des fenêtres-états et leurs interactions.

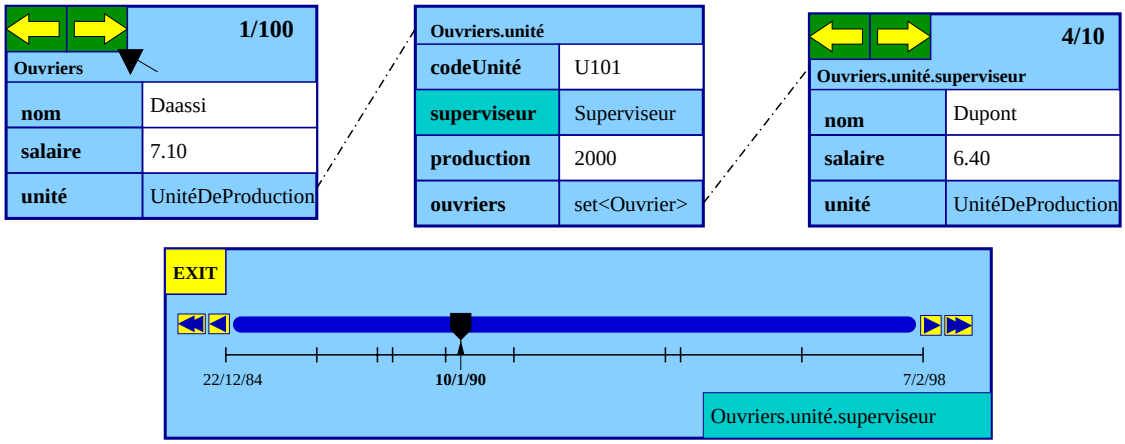
### Visualisation point-par-point de collections d'objets temporels

Jusque làΓnous avons implicitement fait l'hypothèse que l'objet principal ne change pas au cours d'une session.

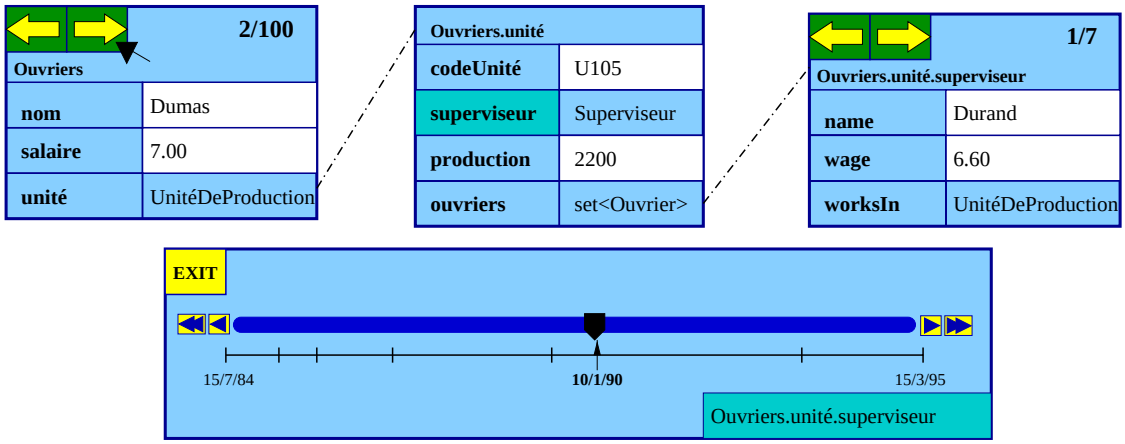
En intégrant la navigation temporelle point-par-point avec la navigation synchrone sur les collections (p. ex. celle définie par PESTO)Γon obtient un paradigme de visualisation dans lequel il est possible de naviguer de façon orthogonaleΓaussi bien au travers des objets temporels composant une collectionΓqu'au travers de leurs associations (temporelles et fugitives)Γ le tout par rapport à un instant de référence fixé. En déplaçant cet instant de référenceΓon peut ensuite "naviguer" sur la composante temporelle des objetsΓtout en respectant cette orthogonalité.

Plus précisémentΓdans la navigation point-par-point sur une collection d'objets temporelsΓla fenêtre-état racine dénote l'état d'un des objets temporels appartenant à la *collection*

*principale*. La fenêtre-état racine comporte alors deux flèches permettant de passer à l'objet suivant ou précédent dans la collection. L'exemple de la figure 4.7 illustre l'effet de l'activation de l'une de ces flèches (en l'occurrence la flèche droite). On remarque au passage sur cette figure qu'il est possible de visualiser des propriétés de type collection (en l'occurrence l'attribut Superviseur::supervise).




(a) État avant l'interaction




(b) État après l'interaction

FIG. 4.7 – *Visualisation temporelle point-par-point: passage d'un objet à un autre au sein de la collection principale, par interaction avec le bouton "Suivant" (étiqueté par une flèche droite) de la fenêtre-état racine.*

Le domaine de navigation temporelle est la couverture des domaines d'observation des objets participant à la collection principale. Lorsque les objets de la collection ne sont pas temporels l'union des domaines temporels de leurs propriétés temporelles tient lieu du domaine d'observation des objets.

# Récapitulatif

Dans ce chapitre nous avons introduit deux interfaces de la plate-forme TEMPOS qui viennent s'ajouter à l'interface de programmation (API) constituée par les types et opérateurs présentés dans le chapitre précédent.

La première de ces interfaces est un langage de requêtes textuel baptisé TEMPOQL. Conçu comme une extension d'OQL TEMPOQL fournit des constructions syntaxiques permettant d'exprimer une large famille de requêtes temporelles de façon claire déclarative et statiquement typée. Par ailleurs suivant l'esprit de l'approche TEMPOS TEMPOQL permet de raisonner sur les historiques en faisant abstraction de leurs représentations. Ce dernier point constitue l'une de ses originalités majeures.

Une autre originalité de TEMPOQL réside dans le concept de "motif d'historiques". Un motif d'historiques est la description en compréhension d'un ensemble éventuellement infini d'historiques au travers d'une expression dans un langage combinant les opérateurs d'OQL avec des variantes d'opérateurs sur des langages réguliers (concaténation répétition avec ou sans contraintes de durée). Ce langage est embarqué dans le modèle TEMPOS sous forme d'un opérateur à valeur booléenne et repris dans la spécification de TEMPOQL.

Enfin l'opérateur de navigation point-par-point au travers d'historiques constitue un troisième point novateur de TEMPOQL. Cet opérateur permet d'appliquer à un objet un chemin de navigation comportant des propriétés de type historique. Par exemple l'expression o.unité.superviseur.salaire permet de retrouver l'historique du salaire du superviseur de l'unité à laquelle est affectée l'ouvrier o. Plus généralement TEMPOQL généralise tous les opérateurs d'OQL aux historiques. Ainsi un opérateur s'appliquant à des arguments de type entier en OQL peut s'appliquer à des arguments de type "historique d'entier" en TEMPOQL.

La deuxième interface que nous avons présentée est une technique de visualisation d'objets temporels dite "point-par-point". L'idée générale de cette technique est de visualiser l'état à un instant de référence d'un ensemble de chemins de navigation partant d'un objet temporel donné. Par ses interactions l'utilisateur peut ensuite modifier soit l'instant de référence soit l'ensemble de chemins de navigation visualisés.

Cette technique de visualisation met l'accent sur la notion de simultanéité. Toutefois elle permet aussi d'observer des changements par le biais de deux opérateurs qui déplacent l'instant de référence jusqu'au prochain (précédent) instant auquel la valeur d'un chemin de navigation change.

Intégrée dans une technique de visualisation de collections d'objets telle que ODEVIEW ou PESTO la visualisation temporelle point-par-point permet de naviguer de façon orthogonale au travers de la dimension temporelle des objets au travers des objets composant une collection et au travers de leurs associations.

# Chapitre 5

# Applications

Au cours de leur développement, les concepts et outils de TEMPOS ont été confrontés à trois types d'applications[1] :

- **Dépouillement d'enquêtes de type "emploi du temps".** En collaboration avec une équipe de recherche en géographie, nous avons étudié une application concernant l'exploitation des résultats d'une enquête sur l'emploi du temps et de l'espace des individus, dans une station touristique de montagne [FCD+98, FCD+99].
- **Gestion de méta-données vidéo.** Dans le but de montrer l'applicabilité de l'approche TEMPOS à des données séquentielles autres que les associations temporelles, nous avons conçu un modèle pour méta-données vidéo fondé sur les concepts de granularité et d'historique [DLF+99a, DLF+00], et nous avons étudié l'adéquation de TEMPOQL à ce contexte.
- **Des variantes temporelles d'applications classiques** du type "suivi des emprunts de documents d'une médiathèque" [DLF+99b] ou encore "suivi du fonctionnement d'une usine". Ces applications ont été choisies de manière à mettre en évidence des aspects de TEMPOS que les autres applications ne permettaient pas d'aborder, comme par exemple la migration d'applications et la navigation au travers d'associations temporelles.

Dans ce chapitre, nous détaillons nos expériences dans le cadre des applications mentionnées dans les deux premiers alinéas ; l'application "suivi du fonctionnement d'une usine" citée dans le troisième alinéa ayant déjà été présentée dans les chapitres précédents.

1. D'autres applications ayant contribué au développement de TEMPOS sont décrites dans [Can97].

## 5.1 Dépouillement d'enquêtes de type emploi du temps

### 5.1.1 Contexte et description de l'application

L'application considérée ici vise à mettre en évidence les fonctionnements d'une station touristique des Alpes Françaises (Valloire) en étudiant ce qui caractérise l'utilisation de l'espace et du temps par les touristes les habitants permanents et les saisonniers.

Du point de vue de la gestion de données l'objet de cette application est d'assister des utilisateurs non-informaticiens dans l'analyse des résultats d'une enquête réalisée auprès de 200 individus. Le protocole de l'enquête est le suivant :

1. Chaque individu interrogé décrit la (ou les) journée(s) précédant l'enquête en répondant systématiquement aux questions : que faites-vous quand où et avec qui ? De façon générale les touristes sont interrogés sur la journée précédant leur participation à l'enquête alors que les saisonniers et les habitants permanents sont questionnés sur les trois jours précédents. Chaque personne interrogée choisit la précision à laquelle il/elle décrit ses mouvements et changements d'activités mais en règle générale on prend comme granularité la minute avec une fourchette d'erreur de 10 minutes.
2. Pour compléter cette information et surtout pour tracer la relation entre l'enchaînement des activités et les déplacements dans l'espace les personnes dessinent sur un fond de carte leurs déplacements et leurs stationnements.
3. Enfin chaque enquête est étayée par des informations sur le profil socioprofessionnel et sur les conditions de séjour des personnes.

L'analyse des données issues de cette enquête vise à répondre à des questions telles que :

- Quels groupes d'individus réalisent quelles activités ?
- Dans quelles conditions les différents groupes d'individus ont-ils la possibilité de réaliser des activités ? Quelles contraintes d'ordre spatial mais aussi temporel existe-t-il dans une station touristique ?
- Comment se forme la chaîne d'activités que les individus réalisent dans l'espace et au cours du temps ? Quelles sont les *trajectoires spatiotemporelles* des individus ?
- Comment et quand se rencontrent les différentes trajectoires individuelles ? Quelles interactions sont mises en jeu entre les divers groupes dans une station touristique ?

L'étude de cette application s'est déroulée dans le cadre du projet MUST [2] regroupant en plus de notre équipe de recherche des chercheurs en géographie du laboratoire SEIGAD [3]

2. MUST: Modélisation des Usages Spatio-Temporels. Projet faisant partie du Programme "Systèmes d'Information Géographique" (PSIG) financé par le CNRS et l'Institut de Géographie National (IGN).

3. SEIGAD: Systèmes Environnementaux, Information Géographiques et Aide à la Décision. Unité de Recherche de l'Université Joseph Fourier de Grenoble

ainsi que des chercheurs en Bases de Données Spatiales et Contraintes de l'INRIA et du CNAM[4]. Une approche complémentaire à celle décrite ici a d'ailleurs été développée par cette dernière équipe (Cf. [GRS98]).

Les travaux du projet MUST partent du constat que les outils existantsΓet en particulier les Systèmes d'Information Géographiques (SIG)Γne répondent pas aux besoins des applications où le temps et l'espace apparaissent entremêlésΓcomme c'est le cas de l'application "Valloire". En effet si l'on peut imaginer d'intégrer dans un SIG du commerce les données relatives aux profils des individus interviewés et les données sur leur position dans l'espace à un instant précisΓil est fort difficile d'adjoindre à ces données les références temporelles permettant de capturer l'évolution de cette position au cours du tempsΓainsi que l'évolution des attributs thématiques variables tels que les activités et les accompagnements des individus.

Pour cerner ces limitationsΓil convient de distinguer les SIG dits *raster* des SIG *vectoriels* [SVP+95]. Les premiers représentent les objets spatiaux sous forme d'une matrice de pixelsΓalors que les seconds utilisent des modes de représentation à base de types géométriques élémentaires tels que les polylignesΓles polygones et les cercles.

Pour modéliser les données de l'application "Valloire" dans un SIG rasterΓchaque itinéraire peut être représenté par une matrice oùΓsur les pixels concernésΓla durée du passage ou du stationnement de l'individu est codée sous forme d'une valeur numérique. Ajouter des attributs qualifiant l'itinéraire (comme l'activité et l'accompagnement en chaque point) revient à augmenter le nombre d'images relatives à chaque interviewéΓchaque pixel d'une matrice ne pouvant être qualifiée que par une seule valeur numérique. Par ailleursΓles SIG raster étant dotés d'opérateurs matriciels assez sophistiquésΓil est possible de comparer des itinéraires spatio-temporels à partir de cette représentationΓmais ceci de façon lourdeΓmalcommode et non généralisableΓau prix d'un effort de programmation considérable.

Du point de vue de la manipulation des données temporellesΓles SIG vectoriels héritent des limitations des SGBD sur lesquels ils s'appuient. En d'autres motsΓlorsque l'on veut observer l'évolution d'un attribut spatial ou thématique dans un SIG vectorielΓson historique doit être codé à partir des types élémentaires (p. ex. le type "entier")Γet la sémantique de ce codage doit être intégrée dans la logique des programmes d'application. Tout comme dans les SIG rasterΓceci rend assez lourd l'effort de programmation nécessaire à l'expression de requêtes.

Les travaux sur la prise en compte du temps dans les SIG sont relativement nombreux (voir p. ex. [CLMS94ΓLMR96ΓPQ96ΓPSZ+97]). Dans la plupart de ces travauxΓl'approche suivie consiste à rajouter dans le modèle de base un type de données modélisant la notion de repère dans le temps (instant ou intervalle) et à l'utiliser pour estampiller des informations géographiques. Les opérateurs associés aux types de données temporelles permettent alors d'exprimer des requêtes spatio-temporelles plus ou moins complexes. Ces requêtes peuvent

4. Plus précisément: projet Verso de l'INRIA et équipe Vertigo du laboratoire CEDRIC/CNAM.

ensuite être exécutées de façon plus ou moins efficace sur des gros volumes de données en s'appuyant sur des techniques d'indexation multi-dimensionnelles telles que les arbres R [Gut84]. Cette approche est développée en particulier dans le cadre du projet TEMPESTA [PQ96]. Malheureusement aucun de ces travaux n'a débouché sur un outil "prêt-à-utiliser".

Ce constat a été la motivation principale de la démarche que nous décrivons dans la suite.

### 5.1.2 Modélisation conceptuelle et logique

Le problème de modélisation des données de l'enquête peut être formulé comme suit : *à un instant donné, une personne seule ou accompagnée effectue une activité dans un lieu ayant une certaine fonction.*

Aussi nous dégageons les ensembles d'entités ci-dessous :

- **Personne** : une personne est identifiable. Il s'agit ici des personnes enquêtées. Pour chaque personne on retient son département d'origine son profil son âge etc.
- **Activité** : les activités possibles sont par exemple déjeuner départ en randonnée etc. L'ensemble des activités est structuré en une hiérarchie de classes qui traduit la classification des activités ("loisirs" est plus général que "sports" qui est à son tour plus général que "golf"). Toutefois afin de simplifier le discours nous n'abordons pas cet aspect par la suite et supposons que les activités sont regroupées au sein d'une seule classe.
- **Accompagnement** : les différents types d'accompagnement sont : famille conjoint enfants amis etc. Dans la mise en oeuvre d'une activité une personne peut être associée à une ou plusieurs de ces valeurs. Par exemple une personne peut être accompagnée de son conjoint et de ses amis ou une personne peut être seulement accompagnée de sa famille.
- **Lieu** : les différents lieux de la station sont géoréférencés (c'est-à-dire accompagnées d'une référence spatiale). Certains sont représentés sur la carte par des points (p. ex. une maison) d'autres par des polygones (p. ex. un centre commercial) ou des polylignes (p. ex. une rue). Chaque lieu est caractérisé par son nom et sa fonction : rues places commerces etc.

Dans la figure 5.1 on trouvera un diagramme de classes des données de l'application. Les associations dont l'évolution est observée (à savoir Activité Accompagnement et Lieu) apparaissent en italique. Ceci afin d'indiquer qu'il ne s'agit pas d'associations simples dans lesquelles un objet d'une des classes participantes est directement associé à un ou plusieurs objets de l'autre classe mais d'*associations temporelles* possédant une structure relativement complexe.

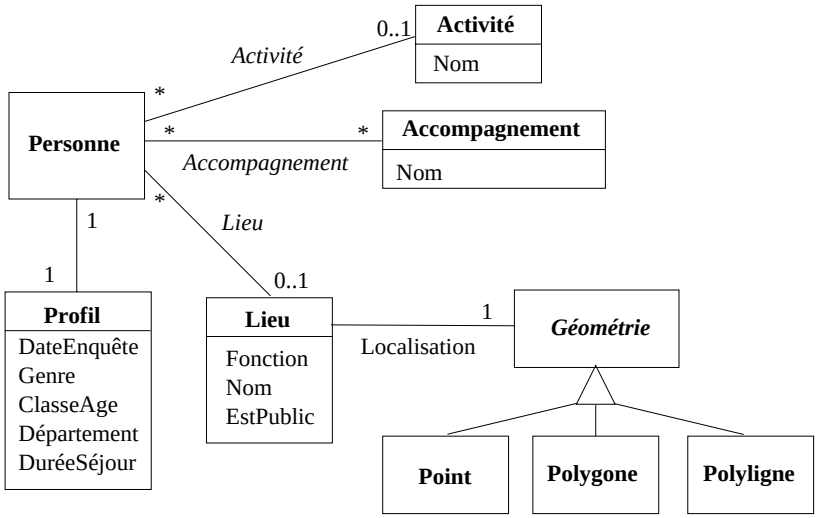



FIG. 5.1 – *Représentation graphique UML*

Pour détailler la structure des associations temporelles (p. ex. l'association Activité)Γnous préconisons l'approche suivante : à chaque instantΓsoit t appartenant à la période sur laquelle porte l'enquêteΓla personne enquêtéeΓsoit pΓeffectue une certaine activitéΓsoit a. Chaque couple <t, a> est appelé un *instantané*. La séquence d'instantanés associée à p décrit l'historique des activités de p.

La figure 5.2 illustre ce principe : la classe Instantané_Activité est décrite par composition des deux classes Instant et Activité. La classe Historique_Activité modélise les historiques d'activités. EnfinΓl'association activités décritΓpour une personne enquêtéeΓles activités qu'elle a effectuées au cours du temps. Le lien inverse est décrit de façon analogue.

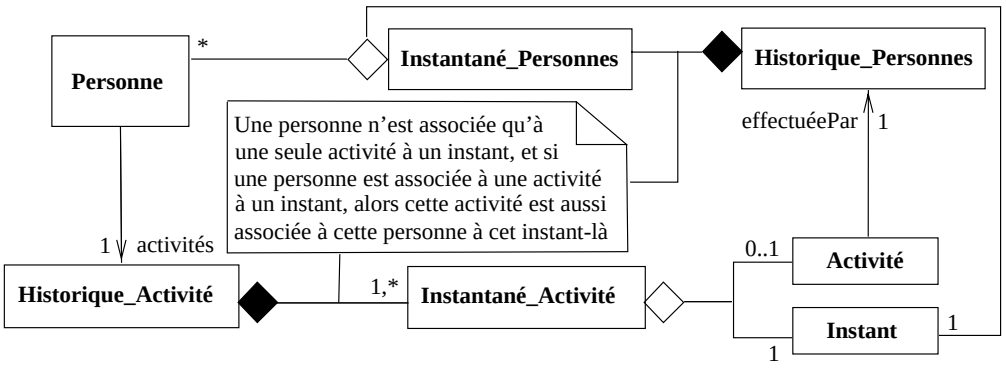



FIG. 5.2 – *L'association temporelle Activité. L'annotation figurant au milieu du schéma exprime l'intégrité référentielle temporelle (Cf. § 3.2.2).*

Cette modélisation des associations temporelles nous semble bien adaptée aux raisonnements intervenant au niveau conceptuelΓcar elle ne privilégie ni le protocole de saisie des informations (dans lequel seuls les instants de changements sont considérés) ni l'efficacité de

la représentation obtenue par traduction directe du schéma résultant vers un système cible (dans le cadre de l'ODMGΓune représentation à base d'intervalles pourrait se révéler plus compacte).

La modélisation des autres associations temporelles s'effectue de manière analogue. Plus généralementΓle diagramme ci-dessus suggère qu'il serait pertinent de définir un stéréotype UML[5] pour dénoter les associations temporelles (cette approche est partiellement explorée dans [TS97]). AlternativementΓle codage des associations temporelles pourrait être intégré dans un patron de conception[6]Γdans la lignée de ceux proposés par [CEF98ΓLan98]. Dans les deux casΓune telle encapsulation du concept d'association temporelle permettrait d'intégrer dans un atelier de génie logiciel donnéΓla connaissance nécessaire à la génération automatique du code correspondant aux associations temporelles figurant dans un diagramme de classesΓ et ce par rapport à différents systèmes cibles (p.ex. l'ODMG ou TEMPOS). La spécification précise d'un tel stéréotype ou patronΓet son intégration dans un atelier de génie logicielΓnous semble une perspective intéressante[7].

À partir de la modélisation conceptuelle ci-dessusΓon obtient le schéma TEMPODL de la figure 5.3. On remarque que lors de la traduction de l'association temporelle Accompagnement Γ seul un chemin de navigation a été retenu (celui qui part de la classe Personne). Ce choix a été effectué au niveau de l'implantationΓcar le chemin inverse s'est révélé inutile pour l'expression de requêtes. En revancheΓles associations temporelles Activité et Lieu sont codées par des propriétés temporelles inverses.

Par ailleursΓil est intéressant d'analyser la portée des choix effectués lors du passage du schéma UML au schéma TEMPODLΓconcernant la modélisation des trajectoires des individus. Plus précisémentΓon observe que pour un individu donnéΓla valeur du chemin de navigation menant de cet objet à sa localisation évolue "en escalier" (c'est-à-dire qu'entre deux instants "effectifs"Γla localisation est supposée constante). Par conséquentΓpour modéliser le déplacement d'un individu le long d'une trajectoireΓon associe une polyligne approchant cette trajectoireΓà chaque instant de l'intervalle de temps pendant lequel le déplacement a lieu. En d'autres termesΓdans la modélisation logique ci-dessusΓsi un individu se déplace d'un point A à un point B pendant un intervalle XΓtout ce que la base de données "sait"Γ c'est qu'à chaque instant dans l'intervalle XΓl'individu se trouve quelque part le long d'une polyligne ayant pour extrémités A et B.

Ce choix répondait bien aux besoins initialement exprimés par les utilisateurs géographesΓ

5. Un stéréotype est un type d'élément de modélisation. La notation UML fournit un certain nombre de stéréotypes prédéfinis (p. ex. interface, processus et flot), et permet l'introduction de nouveaux stéréotypes [FS97].

6. Un patron de conception est une description d'une solution logicielle à un problème spécifique, formulée en des termes suffisamment génériques pour être réutilisable dans un large spectre de contextes.

7. Un travail allant dans ce sens a été réalisé dans le cadre du projet MADS [PSZ+97], en partant d'un modèle entité-relation étendu.

```
class Personne (extent LesPersonnes, key nom) {
  attribute string nom;
  attribute Profil profil;
  valid stepwise granularity Day attribute
        set<Accompagnement> accompagnements;
  valid stepwise granularity Day
        relationship Activité activité
                    inverse Activité::effectueePar;
  valid stepwise granularity Day
     relationship Lieu lieu inverse Lieu::occupants;
}
```

```
class Activité (extent LesActivités) {
  attribute string nom;
  valid stepwise granularity Day
     relationship set<Personne> effectueePar
                    inverse Personne::activité;
}
class Lieu (extent LesLieux, key nom) {
  attribute string nom;
  attribute boolean estPublic;
  valid stepwise granularity Day
        relationship set<Personne> occupants
                    inverse Personne::lieu;
}
```

FIG. 5.3 – *Schéma TempODL de l'application "Valloire". La description des classes qui n'apparaissent pas ici (à savoir Profil et Accompagnement) s'obtiennent par traduction directe du schéma UML figure 5.1.*

car aucune hypothèse sur le mouvement des individus le long des trajectoires n'a été envisagée au départ. CependantΓau fur et à mesure de l'exploitation des donnéesΓil est apparu que l'introduction de telles hypothèsesΓpermettrait d'affiner les résultats de certaines requêtes (notamment celles concernant la recherche de concentrations d'individus dans certaines zones de la station). Par exempleΓil serait naturel de considérer que les individus se déplacent à une vitesse constante le long d'une trajectoire contrainte par le réseau piéton.

MalheureusementΓTEMPOS ne permet pas de modéliser ce type d'interpolation. L'extension de TEMPOS dans cette direction est une perspective intéressanteΓd'autant plus que récemment plusieurs techniques de représentation d'objets mobiles ont été développées [GRS99ΓFGNS00].

### 5.1.3 Expression de requêtes

La première étape dans l'exploitation des données par les utilisateurs géographesΓvisait à produire une image globale de la budgétisation du temps et de l'espace par les individus. Dans ce butΓune dizaine de requêtes permettant d'extraire des vues agrégées des périodes de temps passées par les individus sur chaque activité (respectivement lieu) ont été formulées[8]. Un exemple type de ces requêtes est :

**Q15** : *Quelle est l'activité la plus pratiquée ? C'est-à-dire celle dont la somme du temps passé par les individus à l'effectuer est maximale. Bien entendu, plusieurs activités peuvent répondre à ce critère.*

8. Toutes les requêtes formulées par les utilisateurs l'ont été en langue naturelle ; la traduction vers TEMPOQL étant ensuite effectuée par des informaticiens, au prix de nombreux allers-retours destinés à raffiner la formulation initiale de la requête.

```
/* Type du résultat: bag<Activité> */
define max_temps as /* temps passé par tous les individus à faire l'activité la plus pratiquée. */
        max(select /* temps passé par les individus à faire l'activité a. */
                    sum(select duration(p.activité as x when x = a)
                        from LesPersonnes as p)
                from LesActivités as a);
select a
from LesActivités as a
where sum(select duration(p.activité = a as bool when bool)
                from LesPersonnes as p) = max_temps
```

La deuxième étape de l'analyse des données visait à trouver des routines spatio-temporelles. Pour cela, quelques requêtes ont été formulées dans le but de repérer des moments clés dans la journée, comme par exemple, des instants où un nombre significatif d'individus changent d'activité, de lieu ou d'accompagnement. Cette démarche a permis de construire une structuration de la journée en tranches, représentatives des rythmes journaliers des différentes classes d'individus dans la station.

Concrètement, cette étude s'est basée sur des requêtes dont celle qui suit est un exemple représentatif.

**Q16** : *Retrouver les moments de la journée où au moins un individu a changé d'activité, ainsi que le nombre d'individus qui changent d'activité à ce moment. Ordonner le résultat en ordre décroissant du nombre de changements observés.*

```
/* type du résultat: bag<point_rupture: short, nb_changements: short > */
select point_rupture: point_rupture, nb_changements: count(partition)
from LesPersonnes as p, xchronicle(p.activité) as xs
group by point_rupture: minute_dans_journee(max(xs.time))
/* max(xs.time) est la borne supérieure de l'intervalle xs.time; étant donné que les intervalles d'une
 "XChronique" sont maximaux, cette borne correspond à un instant de changement d'activité. */
order by nb_changements desc
```

Afin d'améliorer la lisibilité de l'expression de cette requête, on a supposé donnée une fonction minute_dans_journee, qui retrouve un entier entre 1 et 1140 (le nombre de minutes dans une journée), correspondant au nombre de minutes écoulées entre l'instant donné en paramètre, et le début de la journée dans laquelle se situe cet instant. Par exemple minute_dans_journee(@"23/12/96 à 1h00") = 60. Cette fonction s'exprime par composition des opérateurs du modèle du temps de TEMPOS (Cf. [Can97]).

À partir des résultats de l'étude précédente, les géographes ont reformulé les requêtes issues de la première étape, en les paramétrant par des instants ou des intervalles. Ceci a

donné lieu à des requêtes telles que : *quelle est l'activité la plus fréquente à midi ?* ou *quel est le lieu plus fréquenté entre 13h et 15h ?*

Enfin, la troisième phase d'analyse s'est orientée vers la recherche de "motifs" de comportement fréquents, et en particulier, de séquencements d'activités ou de déplacements significatifs. Pour satisfaire ce besoin, nous avons formulé des requêtes telles que *quels sont les enchaînements de trois activités les plus fréquents* ou encore :

**Q17 : Recherche d'occurrences de motifs**
*Quelles sont les personnes qui ont enchaîné les activités "ski" et "faire les courses", puis qui ont fait n'importe quelle(s) activité(s) pendant une durée inférieure ou égale à deux heures avant de faire l'activité "repos" ?*

```
/* Type du résultat: bag<Personne> */
select p
from LesPersonnes as p
where p.activité as a matches (a.nom = "ski de piste" or a.nom = "ski de fond"
                               followed by a.nom = "faire les courses"
                               followed by * during <= ♯"2 heures"
                               followed by a.nom = "repos")
```

C'est dans le but de faciliter l'expression de ce type de requêtes qu'a été conçu le langage de recherche de motifs introduit dans le § 4.1.3.

## 5.2 Gestion de méta-données vidéo

Dans cette section, nous présentons les résultats d'une étude visant à unifier des concepts du modèle TEMPOS, avec ceux du modèle VSTORM [LM98, Loz00] pour Bases de Données Vidéo. L'intérêt de cette étude est double :

- Enrichir le spectre d'applications des SGBD Temporels en général, et de TEMPOS en particulier, en étudiant leur applicabilité à la gestion de données séquentielles autres que les associations temporelles proprement dites.
- Fournir un nouvel éclairage sur la modélisation des aspects temporels des données vidéos, en partant des concepts du modèle TEMPOS.

Parmi les diverses problématiques liées à la gestion de documents vidéo, nous nous concentrons sur la modélisation de leurs méta-données, modélisation que nous validons selon le point de vue de deux fonctionnalités : l'interrogation et la composition.

### 5.2.1 Modélisation

Au niveau le plus élémentaire, un document vidéo est composé d'une suite d'images) et d'un flux audio considéré continu, le tout étant destiné à être présenté selon une certaine cadence.

Toutefois, l'information véhiculée par une vidéo ne se restreint pas à ce point de vue. En effet, la perception d'une vidéo par un observateur donne lieu à une interprétation, qui constitue par elle-même une donnée. Cette interprétation peut alors être rattachée à la donnée vidéo elle-même, donnant lieu à la notion de *méta-donnée vidéo* (c.à.d. "donnée sur une donnée").

Un type particulier de méta-données vidéo concerne le partitionnement d'une vidéo en des sous-séquences connexes d'images, selon un critère sémantique. Cela conduit généralement à des structurations arborescentes. Ainsi, dans la cinématographie classique, les films se décomposent en *séquences*, chacune composée de *scènes*, qui elles-mêmes se décomposent en *plans*. D'autres décompositions sont envisageables selon l'intention et le contenu de la vidéo. Par exemple, une vidéo sur un match de tennis se décompose naturellement en *manches*, *jeux* et *points*.

Un autre type de méta-données est constitué de commentaires énoncés par un observateur au sujet de sa perception d'une partie de la vidéo. Nous nous référerons à ce type de méta-données sous le terme d'*annotation*. Le plus souvent, les annotations prennent la forme de mots-clés, mais pour rester général, nous considérons qu'une annotation est n'importe quel objet attaché à une image ou à un segment d'une vidéo.

Ces deux types de méta-données identifiés, nous visons à définir un modèle qui les traite de façon orthogonale, au sens où chaque niveau de structuration puisse être annoté de manière indépendante. Ceci contraste avec des travaux antérieurs (voir plus loin), où seules les images d'une vidéo peuvent être annotées. Cette particularité de notre approche est motivée par le constat suivant : restreindre l'utilisateur à n'annoter que les images d'une vidéo, c'est lui enlever la possibilité d'exprimer des propriétés qui sont vraies au niveau de la scène, sans toutefois être vraies dans chaque image de cette scène (et idem pour les autres niveaux de structuration). Par exemple, le fait que deux personnages discutent dans une scène, est une information qui est vraie d'une scène sans être vraie de chaque image dans cette scène. De façon analogue, dans le cadre d'une vidéo sur un match de tennis, structuré en manches, jeux et points, le fait que le score d'une manche soit de 6–4, n'entraîne rien sur le score d'un jeu donné de cette manche. Le score est donc une annotation qui doit être attachée aux manches et aux jeux de manière indépendante.

Pour modéliser la structuration d'une vidéo, nous exploitons les concepts de ligne de temps et de granularité (Cf. 3.1.1). Plus précisément, nous proposons d'attacher une ligne

de temps à chaque vidéoΓafin de modéliser sa structure “séquencée”. Le nombre de repères temporels dans cette ligne est égal au nombre d'images de la vidéo modélisée. Chaque niveau de structuration d'une vidéo donnée est ensuite modélisé au travers d'une granularité sur la ligne de temps de cette vidéo. AinsiΓchaque scène d'une vidéo est modélisée par un grain. Le niveau de structuration le plus finΓà savoir celui des imagesΓn'échappe pas à ce traitement : l'ensemble d'images d'une vidéo est modélisé par une granularité dont chaque grain est un singleton.

L'ensemble d'annotations attachées aux éléments d'un niveau de structuration d'une vidéo est modélisé au travers d'un historique dont la granularité correspond au niveau de structuration en question. Par exempleΓdans le contexte d'une vidéo annotée sur un match de tennisΓ les scores des jeux se modélisent au travers d'un historique dont la granularité correspond au découpage du match en jeux. Les éléments de l'image de cet historique sont alors des couples d'entiers (p. ex. 6-4 ou 7-6).

Le modèle résultant est schématisé dans la figure 5.4. La vidéo apparaissant sur cette exemple est structurée en plansΓscènes et séquences. Les annotations attachées aux séquences modélisent des activitésΓcelles attachées aux scènes correspondent à des lieuxΓles plans sont annotés par des sous-titresΓet les images par des noms de personnages.

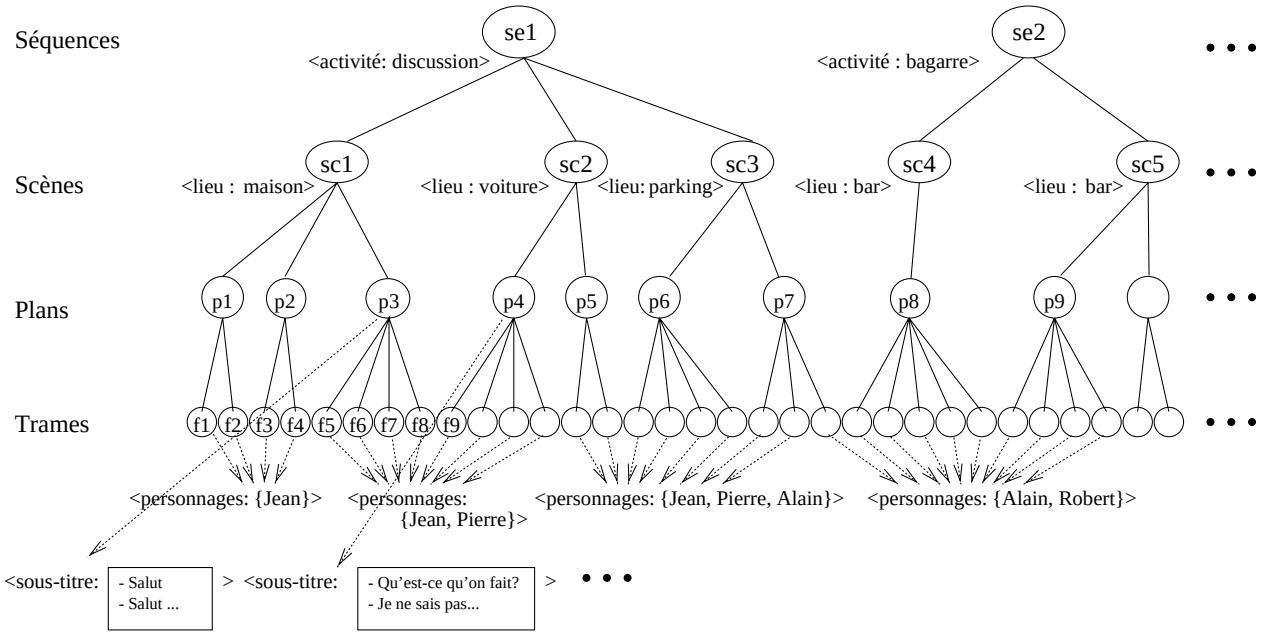



FIG. 5.4 – *Séquences d'annotations attachées à une vidéo.*

L'interface TEMPODL suivante transpose ces idées dans le modèle TEMPOS. Cette interface est destinée à être raffinée pour les besoins de chaque application. AinsiΓune application particulière peutΓsoit rajouter ses propres niveaux de structurationΓsoit raffiner la spécification du type d'annotations qu'elle manipule (dans l'interface ci-dessous le type des

annotations est la racine de la hiérarchie des classesΓsoit Object).

```
interface Video {
  attribute LDT ligneDeTemps;
  attribute Granularité séquences;          /* Granularité modélisant les séquences */
  attribute Granularité scènes;             /* Granularité modélisant les scènes */
  attribute Granularité plans;              /* Granularité modélisant les plans */
  attribute Granularité images;             /* Granularité modélisant les images */
  attribute Historique<Object> annotationsSéquences;
  attribute Historique<Object> annotationsScènes;
  attribute Historique<Object> annotationsPlans;
  attribute Historique<Object> annotationsImages;
  attribute Historique<RéférenceImage> donnéesBrutes;
}
```

Le dernier attribut de cette interface établit le lien entre la partie "méta-données" d'une vidéoΓet ses données proprement dites (appelées ici "données brutes"). La valeur de cet attribut pour une vidéo donnée est un historique de références à des images appartenant à des vidéos brutes. Les vidéos brutes sont modélisées par l'interface définie à cet effet dans VSTORM [Loz00].

```
interface VidéoBrute; /* interface détaillée dans [Loz00] */
interface RéférenceImage {
  attribute VidéoBrute source;
  attribute short postition;
}
```

On remarque que cette modélisation rend possible le partage d'un segment de vidéo brute par plusieurs vidéos annotées. En faitΓla composante "données brutes" d'une vidéo annotée peut être constituée soit de l'intégralité d'une vidéo bruteΓsoit de différentes segments d'une même vidéo bruteΓsoit encore de segments provenant de différentes vidéos brutesΓcomme le schématise la figure 5.5.

Étant donné un objet V appartenant à une classe qui implante l'interface VidéoΓles contraintes suivantes s'imposent :

- V.images = GranularitéMin(V.ligneDeTemps)Γoù GranularitéMin dénote la fonction qui retrouve la granularité la plus fine d'une ligne de tempsΓc'est-à-direΓla granularité dont les grains sont tous des singletons.
- LDT(V.séquences) = LDT(V.scènes) = LDT(V.plans) = V.ligneDeTempsΓoù LDT(G) est la ligne de temps sur laquelle est définie la granularité G (Cf. § 3.1.1).

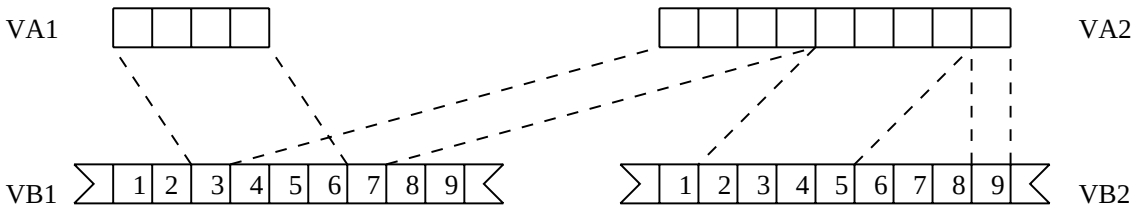



FIG. 5.5 – *Exemple de partage des images d'une vidéo brute par deux vidéos "virtuelles". VA1 et VA2 dénotent les historiques des références à des images associés à deux vidéos annotées. VB1 et VB2 dénotent les images de deux vidéos brutes.*

- V.plans $\prec$ V.scènes $\prec$ V.séquences
- Granularité(Domaine(V.annotationsSéquences)) = V.sequences
- Granularité(Domaine(V.annotationsScènes)) = V.scènes
- Granularité(Domaine(V.annotationsPlans)) = V.plans
- Granularité(Domaine(V.annotationsImages) = V.images
- Granularité(Domaine(V.donnéesBrutes)) = V.images

La gestion de ces contraintes est à la charge des programmes d'applicationΓcar l'ODMG ne permet pas la spécification de contraintes d'intégrité au niveau du schéma.

### 5.2.2 Interrogation

Dans le but d'illustrer l'adéquation des opérateurs de TEMPOQL à l'interrogation de vidéos annotéesΓnous considérons une base constituée d'une collection de films. Chaque filmΓ identifié par son titreΓest qualifié par son réalisateurΓses acteursΓses personnagesΓet une correspondance entre acteurs et personnages. Les images de chaque film sont annotées par les personnages qui y apparaissentΓet les scènes par une description du lieu où l'action se déroule (comme p. ex. "maison"Γ"parking"Γetc.). La classe suivanteΓqui implante l'interface Vidéo définie ci-dessusΓmodélise le concept de film.

```
class Film: Vidéo (extent LesFilms, key titre) {
  attribute string titre;
  attribute string réalisateur;
  attribute set<String> acteurs; /* Tous les acteurs qui apparaissent dans le film */
  attribute set<String> personnages; /* Tous les personnages qui apparaissent dans le film */
  attribute Historique<string> annotationsScènes; /* La valeur de cet historique pour une scène
                                                     donnée, dénote un lieu */
  attribute Historique<set<string>> annotationsImages; /* Chaque valeur de cet historique dénote
                                                          un ensemble de personnages */
```

```
    /* La correspondance entre les acteurs et les personnages d'un film est encapsulée dans les deux
    méthodes suivantes. On suppose pour simplifier que chaque acteur joue un seul personnage, et
    que chaque personnage est joué par un seul acteur. */
    string personnage(string acteur);
    string acteur(string personnage);
}
```

Les deux premières requêtes que nous considérons illustrent l'utilité des opérateurs de restriction vis-à-vis de la manipulation d'annotations. Dans la première requête l'historique d'annotations attachées aux imagmes d'un film est restreint à un intervalle donné. Dans la deuxième requête l'historique d'annotations de chaque film est restreint aux images où sa valeur satisfait une condition.

**Q18 : Restriction (selon le domaine)**

*Retrouver les noms des personnages qui apparaissent au moins une fois au cours des 20 premières secondes du film intitulé "Espoirs". On suppose que la cadence de présentation naturelle de ce film est de 30 images/seconde*[9].

```
/* type du résultat : bag<string> – chaque chaîne dénote un personnage. */
flatten(select ensPersonnages
        from TheMovies as F,
             range(F.personnages during [ 0 @ F.images | (20 * 30) # F.images ]) as anActorSet
        where F.titre = "Espoir")
/* Rappel: N @ G (resp. N # G) est un instant (resp. une durée) exprimé(e) à la granularité G.
[I | D] dénote un intervalle dont la borne inférieure est I et la durée est D. */
```

**Q19 : Restriction (selon l'image).**

*Dans quels films "Jean Gabin" apparaît pendant au moins 15 minutes, en supposant que la cadence de présentation naturelle de chaque film est de 30 images/seconde.*

```
/* type du résultat : bag<Film> */
select F
from LesFilms as F
where duration(F.annotationsImages as EP when F.personnage("Jean Gabin") in EP)
      >= (15 * 60 * 30) # F.frames
/* F.Personnage("Jean Gabin") est le personnage interprété par Jean Gabin. */
```

Dans ces deux requêtes nous faisons l'hypothèse que la cadence de présentation naturelle de chaque film est constante. Cependant les videos annotées peuvent être formées de segments

9. La cadence de présentation naturelle d'une vidéo est la cadence à laquelle cette vidéo a été tournée. Si la vidéo est présentée plus vite (resp. plus lentement) que sa cadence naturelle, l'observateur perçoit la présentation comme étant accélérée (resp. ralentie).

provenant de différentes videos brutes, chacune possédant sa propre cadence de présentation naturelle. Il s'en suit que la fonction de conversion entre une durée exprimée en secondes, et une durée exprimée en nombre d'images, ne s'exprime pas par une simple multiplication. Cet aspect de l'interrogation de données vidéo, qui à notre connaissance n'est pas abordée par les travaux existants [10], met en évidence le besoin d'intégrer en TEMPOS, la notion de fonction de conversion entre granularités définies sur des lignes de temps différentes.

L'opérateur de regroupement selon une granularité fournit un mécanisme pour agréger les annotations attachées à un niveau donné, vers des niveaux supérieurs. Ceci permet de raisonner à différents niveaux de structuration d'une vidéo au sein d'une une même requête.

**Q20 : Regroupement selon une granularité**

*Retrouver les scènes du film "Espoirs" où le personnage "Jean" apparaît dans au moins la moitié des images de cette scène.*

```
/* type du résultat : EDI – dont la granularité dénote le découpage d'un film en scènes. */
element(
select domain(map partition
              on F.annotationsImages
              group by F.scènes
              having duration(partition as EP when "Jean" in EP) >= duration(partition) / 2)
from LesFilms as F
where F.title = "Espoirs")
```

Composé avec l'opérateur de jointure temporelle, l'opérateur de regroupement selon une granularité permet aussi d'exprimer des requêtes faisant intervenir des annotations attachées à différents niveaux, comme par exemple *"retrouver les scènes dans un film donné, qui ont lieu dans une maison, et dans laquelle le personnage Jean est présent dans au moins une image"*.

Enfin, les opérateurs de découpage d'historiques et le langage de description de motifs, permettent de raisonner sur la structure séquencée des vidéos, et plus précisément, d'exprimer des requêtes telles que *"étant donnée une vidéo, quels sont les personnages qui n'apparaissent pas avant la première parution d'un personnage donné"*, ou encore *"retrouver les séquences dans lesquelles un personnage donné apparaît pendant plus de trois secondes, puis disparaît pendant au moins dix secondes, avant de réapparaître"*.

10. En effet, les langages de requêtes pour données vidéo qui permettent l'expression de durées en termes de nombre de secondes, le font sous l'hypothèse d'une cadence de présentation constante (voir par exemple [DC98]).

### 5.2.3 Comparaison avec des approches existantes

Les spécificités des applications manipulant des données vidéo ont été étudiées sous diverses perspectives par un grand nombre de travaux (Cf. [Loz00] pour un aperçu général). Une partie importante d'entre eux se concentrent sur des aspects "systèmes" tels que le stockage, le transfert en temps réel dans un environnement distribué, la qualité de la présentation en la présence de ressources matérielles réduites, etc. D'autres se consacrent à l'analyse du contenu de la vidéo : reconnaissance de formes et de mouvements, segmentation automatique d'une vidéo en plans, etc. Enfin, une troisième catégorie de travaux abordent des aspects liés à la représentation et la manipulation du contenu sémantique des vidéos (c.à.d. leurs méta-données), en faisant abstraction de la manière dont celui-ci est extrait. C'est dans ce dernier contexte que se place notre contribution.

De nombreux formats semi-structurés pour la représentation des méta-données vidéo ont été proposés, et des efforts de standardisation dans ce domaine, notamment MPEG-7 [NL99], sont en passe d'aboutir. Dans son état actuel, MPEG-7 modélise les méta-données d'une vidéo au travers de *descripteurs* : entités hiérarchiquement liées, chacune pouvant être rattachée à tout ou une partie d'un document vidéo. MPEG-7 est conçu comme un format flexible, au sens où l'utilisateur a la possibilité de définir ses propres types de descripteurs, en plus de ceux fournis par le standard. Le jeu de descripteurs fournis par le standard est très vaste. Il inclut notamment des descripteurs permettant de définir des découpages d'une vidéo (et a fortiori de spécifier sa structuration), ainsi que des descripteurs permettant d'indexer son contenu sémantique au travers d'objets et d'événements.

Notre contribution est complémentaire à celles de MPEG-7, au sens où nous ne visons pas à définir un format de stockage et/ou d'échange de méta-données vidéo, mais plutôt un modèle de représentation des aspects "temporels" de ces méta-données, adapté à leur interrogation et à leur édition dans le contexte d'une Base de Données.

Ceci nous amène à placer notre contribution dans le cadre des modèles et langages pour Bases de Données Vidéos, domaine dans lequel il existe des nombreuses propositions [EJA+97]. Par rapport à ces propositions, le point innovant de notre approche réside dans l'orthogonalité avec laquelle la structuration logique et les annotations d'une vidéo sont traitées. En effet, dans les modèles de vidéos annotées qui reconnaissent la notion de structuration logique, (p. ex. [GR94], [HMS95] et [RKLL98]), les annotations ne peuvent être attachées qu'aux éléments du niveau le plus bas de structuration, à savoir les images[11]. Pire encore, dans des modèles tels qu'AVIS [ACC+96] et CVOT [LGOS97], la structuration est dérivée à partir des annotations, c'est-à-dire qu'à partir du moment où une annotation est attachée à un intervalle

11. Dans [HMS95], les annotations peuvent être attachées à n'importe quel intervalle de images, mais la sémantique de cette association est la même que si l'annotation était attachée à chaque image dans cet intervalle.

d'imagesΓcet intervalle devient un élément de la structuration. Il s'en suit que l'utilisateur n'a pas la possibilité de définir une structuration sur une vidéo sans y inclure des annotations.

[HMS95] est l'un des travaux les plus similaires au nôtre. Les auteurs décrivent un cadre général pour la modélisation de vidéosΓbasé sur le concept de *flot* : suite ordonnée de références à des données brutes dénotant des images et des échantillons de sons. Les annotations sont attachées à des intervalles sur ces flotsΓet les requêtes temporelles sont formulées en utilisant cette représentation à base d'intervalles. Contrairement à TempOQLΓqui est une extension conservatrice d'un langage de requêtes standardΓle langage de consultation développé dans [HMS95] possède une syntaxe propreΓbien qu'inspirée du bloc *select/from/where* de SQL. Cette dernière remarque s'applique également à la plupart de langages de requêtes pour vidéos annotéesΓcomme par exemple VideoSQL [OT93] and VIQS [HS95].

Un autre travail similaire au notre dans sa démarche est CVOT [LGOS97]Γoù il est proposé d'utiliser le concept d'historique tel que défini dans le modèle TIGUKAT [GO93]Γpour modéliser la suite d'annotations d'une vidéo. Les requêtes temporelles sur ces annotations s'expriment alors par composition des opérateurs sur le type historique. OrΓce jeu d'opérateurs est très restreint : il est composé essentiellement de deux opérateurs de restriction (selon le domaine et selon l'image)Γet d'un opérateur de transformation d'un historique en sa représentation par intervalles (Cf. 3.1.2). En conséquenceΓl'expression de requêtes temporelles se fait essentiellement en raisonnant sur une représentation par intervalles des historiquesΓtout comme dans [HMS95]. Par ailleursΓ[LGOS97] ne prend pas en compte le caractère relatif du temps au sein d'une vidéo. En effetΓchaque annotation est associée à une date appartenant à un repère temporel "global"Γcomme par exemple *"le 1/1/1900 à 1:00:00"*. Ceci contraste avec notre approcheΓdans laquelle chaque vidéo possède son propre repère temporel (c.à.d. sa propre ligne de temps).

Cette idée d'associer un repère temporel à chaque historique a été introduite dans [SLR96]Γ travail qui se place dans le contexte des applications manipulant des séries chronologiques (Cf. § 2.3.2). Le modèle proposé dans [SLR96] repose sur un type de données abstrait (TAD) appelé *séquence*Γqui modélise des fonctions à domaine sur un ensemble d'entiers. Ce TAD est muni d'opérateurs algébriques tels que la restriction (selon le domaine et selon l'image)Γ le produit interneΓet le regroupement (selon une granularité et selon une durée). ToutefoisΓà la différence de notre approcheΓ[SLR96] ne considère que des représentations de séquences à base d'instantsΓau détriment des représentations par intervalles quiΓcomme nous l'avons vu précédemmentΓse révèlent plus adaptées vis-à-vis des raisonnements sur les changements.

EnfinΓ l'idée de faire partager des segments de "vidéos brutes" par plusieurs "vidéos annotées" (dites aussi "vidéos virtuelles") se retrouve dans un bon nombre de modèles pour données vidéoΓdont [OT93]Γ[HMS95]Γ[WDG95] et [LM98].

## Récapitulatif

Nous avons étudié l'utilisation de la plate-forme TEMPOS dans le contexte de deux applications : dépouillement d'enquêtes de type "emploi du temps" et gestion de vidéos annotées.

La première de ces applications a fait l'objet d'un projet de recherche faisant intervenir des chercheurs en géographie. Les données que nous avons utilisées dans cette expérimentation sont issues d'une enquête conduite dans la station touristique Valloire des Alpes Françaises. Les requêtes que nous avons été amenés à formuler l'ont été en réponse à des réels besoins de la part des géographes. Les résultats de ces requêtes ont d'ailleurs été exploités dans une analyse ayant conduit à des propositions concrètes pour le réaménagement de la station concernée [Cha99]. Cette expérimentation a beaucoup contribué au développement du modèle TEMPOS notamment en mettant en évidence le besoin d'un opérateur de recherche de motifs d'historiques. Elle a aussi permis de mettre en avant les limites de l'approche TEMPOS en ce qui concerne la modélisation de trajectoires.

L'application de TEMPOS à la gestion de vidéos est le résultat d'un travail de réflexion mené en collaboration avec des chercheurs en Bases de Données Vidéo (modèle VSTORM [Loz00]). Ce travail a débouché sur des apports mutuels. D'une part le modèle TEMPOS a été enrichi au cours de cette réflexion notamment par l'introduction du concept de "ligne de temps". D'autre part cette étude a permis de mettre en évidence le besoin de rendre orthogonaux les aspects "structuration" et "annotation" des méta-données vidéo ce qui a contribué à clarifier certains choix et concepts du modèle VSTORM. Un résultat de cette collaboration que nous n'abordons pas dans cette thèse (car sortant de son cadre) concerne la création et composition de vidéos annotées. Dans [DLF+99a] nous avons étendu les opérateurs de composition de vidéos proposés dans [HMS95] et [WDG95][12] de manière à préserver les méta-données des vidéos lors du processus de composition. En effet dans [HMS95] et [WDG95] les vidéos créées par composition n'ont par défaut aucune structuration ni annotation conduisant ainsi à une perte d'information. En revanche les opérateurs de composition que nous avons proposés déduisent une structuration et des annotations pour la vidéo résultante par restriction ou combinaison de celles des vidéos composées.

12. Ces opérateurs sont au nombre de cinq : extraction d'un segment d'une vidéo, concaténation de deux vidéos, intersection, union et différence.

# Chapitre 6

# Implantation

La plupart des concepts et fonctionnalités de TEMPOS ont été implantés sous forme d'un prototype au dessus du SGBD à objets $O_2$ [BDK92]. Ce prototype est essentiellement constitué d'une bibliothèque de classes correspondant aux types introduits dans le chapitre 3, de deux préprocesseurs implantant TEMPODL et TEMPOQL, et d'un outil de visualisation interactive, conçue selon la technique de navigation point par point.

La figure 6.1 décrit l'architecture du prototype.

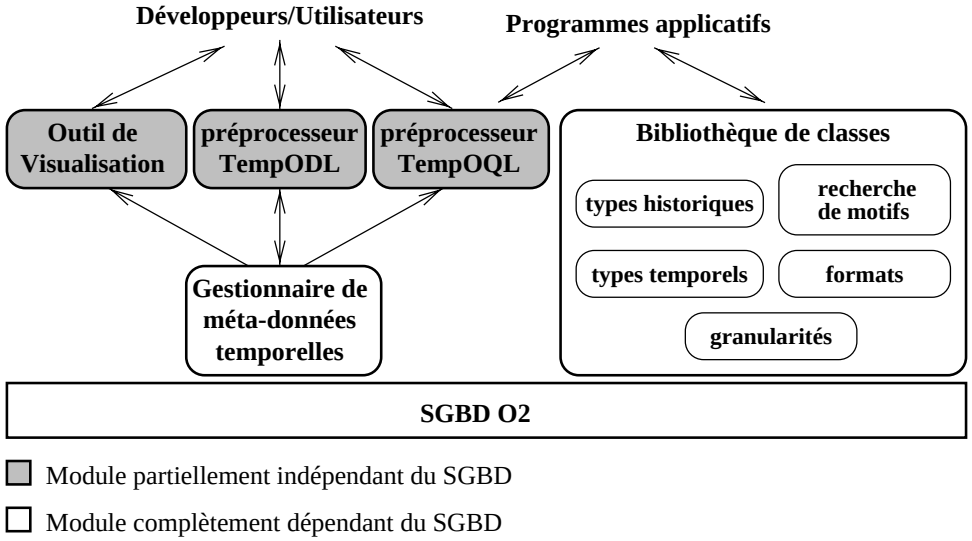



FIG. 6.1 – *Architecture du prototype.*

Dans cette architecture, il est fait référence à un "gestionnaire de méta-données temporelles". Ce module est chargé de tracer les déclarations de classes et de propriétés temporelles effectuées au travers du préprocesseur TEMPODL, et de rendre cette information accessible au préprocesseur TEMPOQL et à l'outil de visualisation. En effet, ces deux modules requièrent une connaissance précise du schéma de la base sur laquelle ils opèrent (Cf § 6.2.1 et § 6.2.3 pour plus de détails).

## 6.1 Implantation du modèle

La bibliothèque de classes qui implante le modèle de données TEMPOS est codée dans le langage de programmation $O_2C$ du SGBD $O_2$. $O_2C$ est une extension à objets du langage C dans laquelle les objets et variables persistantes sont manipulées de façon transparente c'est-à-dire au même titre que les objets et variables non-persistantes. Le choix d'$O_2C$ a été guidé d'une part par sa simplicité et d'autre part parce que les classes ainsi codées peuvent être exportées facilement vers les autres langages de programmation supportés par le SGDB $O_2$ (à savoir C++, Java et SmallTalk) alors qu'il est très difficile d'invoquer des méthodes utilisant la passerelle C++ pour $O_2$ depuis la passerelle Java et réciproquement. Par contre $O_2C$ possède le désavantage d'être un langage propriétaire supporté uniquement par le système $O_2$.

Les classes composant la bibliothèque sont organisées en cinq modules (Cf. figure 6.1). Les modules nommés "types temporels", "formats" et "granularités" implantent tous les types du modèle du temps (Cf § 3.1.1). Leur développement s'inscrit dans un travail antérieur à notre projet de thèse [CSF96, CD97, Can97]. Nous ne discuterons de leur implantation que pour montrer comment le concept de "ligne de temps" est pris en compte (ce concept étant absent des premières versions de TEMPOS) et pour expliciter certains choix qui ne sont pas abordés dans les références ci-dessus.

Le module "types historiques" regroupe les implantations du type paramétré Historique et des types modélisant les concepts de classe et de propriété temporelle. Ce module a aussi fait l'objet d'une implantation préliminaire décrite dans [Can97], implantation revue et complétée au cours de notre projet de thèse.

Enfin, le module "recherche de motifs" fournit une implantation du langage décrit dans le § 4.1.3 et de l'opérateur matches du type historique. Nous reviendrons sur ce module dans la section suivante.

### 6.1.1 Représentation des TAD

Dans les paragraphes suivants, nous présentons certaines représentations possibles des TAD introduits dans le § 3.1.1.

**Ligne de temps.** Une ligne de temps est un ensemble fini de "repères temporels", muni d'un ordre linéaire et total. Un tel ensemble peut être représenté par n'importe quel intervalle d'entiers de cardinalité égale au nombre de repères temporels. Si on fixe par convention à zéro la borne inférieure des intervalles dénotant des lignes de temps, alors une ligne de temps se représente simplement par un entier positif.

**Granularité.** Une granularité est une partition d'une ligne de temps en des sous-ensembles convexes. En assimilant une ligne de temps à un intervalle d'entiers, une granularité peut se représenter par une suite de N−1 entiers positifs, N étant le nombre de grains de la granularité concernée. Chacun de ces entiers représente la borne supérieure de l'un des éléments de la partition (c.à.d. d'un grain). La borne supérieure du dernier grain dans la granularité n'est pas explicitement représentée, car nécessairement égale à la borne supérieure de la ligne de temps. À titre d'exemple, soit une ligne de temps représentée par l'intervalle [0, 39], la liste suivante représente une granularité sur cette ligne de temps : [4,13,24,29,35]. Cette granularité comporte 6 grains.

Du point de vue de l'espace occupé par les données, il est souvent intéressant de considérer des représentations des granularités intégrant le concept de "périodicité". Par exemple, la granularité "Année" correspond à une partition d'une ligne de temps en des périodes de 365 ou 366 jours, avec des fortes régularités sur les tailles des périodes. Toutefois, la spécification et la représentation de ces ensembles périodiques est dans le cas général, une tâche assez complexe [LMF86]. Pour contourner cette complexité, [Can97] propose d'associer une classe à chaque granularité. Le développeur d'une classe correspondant à une granularité est alors responsable de fournir une représentation de la granularité en question, et d'implanter les algorithmes de conversion entre cette granularité et les autres granularités définies sur la même ligne de temps. Le prototype TEMPOS fournit des classes correspondant aux granularités usuelles du calendrier grégorien (Année, Mois, Semaine, Jour, Heure, Minute et Seconde) pour un ligne de temps s'étalant du 1/1/1900 à 0h00:00 au 31/12/2099 à 23h59:59.

**Instants et durées.** Dans le prototype TEMPOS, instants et durées sont décrits au travers d'une paire composée d'un entier positif et d'une granularité. Ceci contraste avec de nombreux systèmes existants, dans lesquels les instants sont représentés par leur forme multigranulaire dans un système de granularités. Par exemple, une date est souvent représentée par trois entiers dénotant l'année, le mois dans l'année, et le quantième dans le mois. La référence [DS95] fournit un aperçu comparatif des modes de représentation d'instants utilisés dans les SGBD commerciaux.

**Intervalles et EDI.** Un intervalle peut être représenté, soit au travers de deux couples d'instants (choix retenu dans l'implantation de TEMPOS), soit par un instant (sa borne inférieure) et une durée. Alternativement, vu que la borne inférieure et la borne supérieure d'un intervalle ont toutes les deux la même granularité, il est envisageable de représenter un intervalle par une granularité et deux entiers positifs (les positions des bornes).

De façon similaire, un EDI peut être représenté soit par une liste d'instants, soit par un couple composé d'une granularité et d'une liste d'entiers positifs. Du point de vue de la l'espace mémoire utilisé, il est intéressant dans certains cas de regrouper plusieurs entiers

consécutifs d'un EDI au sein d'un intervalle. Ceci conduit à une représentation sous forme d'un couple composé d'une granularité et d'une collection d'intervallesΓintervalles qui à leur tour sont représentés par un couple d'entiers.

À titre d'exempleΓl'EDI contenant les instants listés ci-dessous :

@"1/1/98", @"2/1/98", @"3/1/98", @"4/1/98", @"5/1/98", @"6/1/98",
@"7/1/98", @"12/1/98", @"13/1/98", @"14/1/98", @"15/1/98"

peut se représenter des deux façons suivantes :

(1) ⟨ Jour, [35794, 35795, 35796, 35797, 35798, 35799, 35800, 35805, 35806, 35807, 35808] ⟩
(2) ⟨ Jour, [⟨ 35794, 35800 ⟩, ⟨ 35805, 35808 ⟩] ⟩.

TEMPOS fournit une implantation du type EDI pour chacune de ces deux représentations. Plus précisémentΓle type abstrait EDI est modélisé au travers d'une classe virtuelle[1] appelée TSéquence. Cette classe virtuelle se spécialise en deux classes "réelles"Γà savoir ISéquence (correspondant à la représentation par instants) et XSéquence (correspondant à celle par intervalles). Les méthodes correspondant aux opérateurs sur les EDI ne sont implantées qu'au niveau de ces deux dernières classes. On remarque que lorsqueΓdans le cadre d'un programme ou d'une requêteΓon invoque une méthode sur une expression dont le type statique est TSéquenceΓla liaison entre le nom de la méthode et son implantation est effectuée par rapport au type exact du receveurΓqui n'est connu qu'au moment de l'exécution du programme (mécanisme de liaison dynamique des langages à objets).

**Historiques.** Dans le § 3.1.2Γnous avons introduit deux modes de représentation des historiques : les IChroniques et les XChroniques. [Can97] considère en plus de ces deuxΓun autre mode de représentation à base d'ensemble d'instants : les DChroniques. Plus précisémentΓune DChronique est un ensemble de couples ⟨ EDIΓvaleur ⟩Γchaque EDI pouvant être représenté par l'une quelconque des méthodes énumérées ci-dessus. L'historique pris comme exemple dans la page 47 pour introduire les IChroniques et les XChroniquesΓest représenté par la DChronique suivante :

{ ⟨{1, 2, 4, 9, 10}, v1⟩, ⟨{5, 6, 7}, v2⟩, ⟨{8}, v3⟩ }.

Le choix entre ces trois représentations dépend des caractéristiques de l'historique concerné. Par exempleΓsi l'historique comporte beaucoup de répétitions de valeurs successives (c.à.d. la valeur à l'instant i et à l'instant i+1 coïncident dans de nombreux cas)Γalors une représentation par XChronique est assez naturelle. De même si la cardinalité de l'image de l'historique

1. On dit qu'une classe est "virtuelle" en $O_2$, si elle n'implante pas toutes les méthodes présentes dans sa spécification. Les classes virtuelles ne sont pas destinées à être instanciées.

est faible par rapport à la cardinalité de son domaine, alors une représentation à base de DChronique peut se révéler moins coûteuse en espace. En tout cas, TEMPOS fournit une implantation du type historique pour chacune de ces trois représentations. En d'autres mots, le TAD historique est modélisé par une classe virtuelle appelée Chronique, qui se spécialise en trois classes "réelles" : IChronique, XChronique et DChronique. Comme dans le cas des TSéquences, la liaison entre un nom de méthode sur la classe Historique et son implantation est effectuée dynamiquement lors de l'évaluation d'une requête ou de l'exécution du programme.

Dans certains cas, on peut envisager d'autres représentations. Par exemple, dans le cas d'un historique dont le domaine est un intervalle (ou plus généralement n'importe quelle séquence périodique d'instants), il est intéressant de considérer des représentations de la forme ⟨ Intervalle, liste de valeurs ⟩, où la cardinalité de l'intervalle et celle de la liste coïncident, comme par exemple : ⟨ [1..6], [v1, v2, v1, v3, v4, v5] ⟩.

Enfin, lorsque les éléments de l'image d'un historique sont des ensembles, il est possible de représenter cet historique en associant des intervalles aux éléments apparaissant dans ces ensembles. Par exemple, en supposant que dans la DChronique ci-dessus v1 = {a1, a2}, v2 = {a3, a4} and v3 = {a1, a3}, alors une représentation possible de l'historique concerné est :

{ ([1..2], a1), ([1..2], a2), ([4..4], a1), ([4..4], a2) ([5..7], a4), ([5..8], a3) ([8..10], a1), ([9..10], a2) }

### 6.1.2 Prise en compte de la paramétrisation du TAD Historique

L'une des difficultés que nous avons rencontrées lors de l'implantation de la bibliothèque de classes concerne l'absence du concept de *classe paramétrée* dans le modèle de données du SGBD $O_2$. En effet, le type historique se traduit naturellement en une telle classe.

Une des solutions que nous avons envisagées consiste à générer une classe pour chaque instanciation du type historique requise par une application donnée. Dans le cas de l'application "Usine" présentée dans le chapitre 3, ceci revient à générer une classe Historique_Superviseur pour les historiques de superviseurs, Historique_float pour les historiques de flottants, etc. Dans des situations réalistes, cette approche conduit rapidement à une explosion du nombre de classes gérées par le SGBD. De plus, les signatures de certains opérateurs sur les historiques (plus précisément les jointures) ne peuvent pas être codés fidèlement dans cette approche, car elles sont génératrices d'un ensemble infini d'instanciations du type historique. Pour illustrer ce point, nous rappelons partiellement la spécification du type historique en ODMG (augmenté des interfaces paramétrées) :

```
interface Historique<T> {
  Historique<struct<gauche: T, droite: Object>> ProdI(Historique<Object>);
  Historique<struct<gauche: T, droite: Object>> ProdE(Historique<Object>);
   ...
}
```

Supposons maintenant que cette interface soit instanciée avec T = Employé. Le type du résultat de la méthode ProdI est alors Historique<struct<gauche: Employé, droite: Object>>Γce qui correspond à une instanciation de l'interface Historique avec T = struct<gauche: Employé, droite: Object>Γinstanciation qui engendre à son tour une autre instanciation et ainsi récursivement. Bien entenduΓles programmes applicatifs ne requièrent pas que toutes ces instanciations soient effectivement réalisées. Mais pour déterminer lesquelles de ces instanciations sont requisesΓil faut effectuer une analyse du code des programmes applicatifs concernés.

Une deuxième solution consiste à coder toutes les instanciations possibles du type historique au travers d'une seule classe (que nous appelons Historique)Γmodélisant des historiques dont les éléments de l'image sont de type Object. Au détriment du typage statiqueΓune telle classe peut alors être utilisée pour modéliser des historiques de n'importe quelle classe. Cette approche est souvent adoptée dans les langages de programmation à objets ne disposant pas du concept de classe paramétrée (p. ex. Java)Γpour coder le concept de "collection d'objets".

Dans le cas spécifique du modèle de données d'$O_2$Γcette approche pose un problème supplémentaire. En effetΓ$O_2$ fait une distinction très forte entre les types dits *littéraux* (entiersΓ réelsΓflottantsΓetc.)Γet les types *mutables* (toutes les instances des classes héritant de la classe Object). OrΓle principe de substitutionΓessentiel vis-à-vis du codage des historiques que nous adoptonsΓne s'applique pas entre types littéraux et types mutables (p. ex. un entier ne peut pas prendre la place d'une instance de la classe Object). DoncΓla classe Historique ne peut pas être utilisée directement pour coder des historiques d'entiers.

Pour faire face à ce problèmeΓnous avons introduit une classe pour chacun des types littérauxΓc'est-à-dire une classe Integer pour les entiersΓReal pour les réelsΓString pour les chaînesΓetc. Ces classes héritent de la classe ObjectΓet il est par conséquent possible de coder des historiques d'instances de type IntegerΓRealΓStringΓetc. au travers de la classe Historique. La même approche est utilisée pour coder les types "n-uplet" (struct) et "collection" (listΓset et bag)Γqui sont aussi des types littéraux en $O_2$

Le désavantage de cette approche est assez évident : chaque fois que la valeur d'un historique à un instant donné est extraite (au travers de la méthode VS de l'interface Historique[2]Γ une coercition[3] explicite de son type doit être introduite. En d'autres motsΓpour accéder à l'objet dénotant le superviseur d'un ouvrier O par rapport à l'instant 1/1/1998Γil faut écrire une expression de la forme :

```
((UnitéDeProduction) O.unité.Valeur(Instant("1/1/1998"))).superviseur
```
[4]

2. Une remarque similaire s'applique vis-à-vis de la méthode Image.

3. Traduction du terme anglo-saxon *casting*.

4. Le constructeur de la classe Instant prend en entrée une chaîne de caractères et l'interprète par rapport au système de formats par défaut (Cf. § 4.1.1). La méthode Valeur extrait la valeur d'un historique à un instant.

Pire encore, pour obtenir le salaire d'un ouvrier O au 1/1/1998, on doit écrire :

```
((Integer)O.salaire.Valeur(Instant("1/1/1998"))).literal
```

Pour contourner cet inconvénient, le préprocesseur TEMPOQL que nous présentons plus loin, est conçu de manière à insérer ces coercitions dans le code des requêtes.

### 6.1.3 Les classes et les propriétés temporelles

#### Attributs temporels

L'implantation du concept d'attribut temporel en TEMPOS est basée sur une classe nommée VariableTemporelle. Un attribut temporel est codé par un attribut (au sens du modèle $O_2$), dont le type est VariableTemporelle, ou l'un de ses sous-types.

La classe VariableTemporelle se spécialise en VariableDeValidite et VariableDeTransaction. La classe VariableDeValidite se spécialise à son tour en VariableEscalier, VariableDiscrete, et VariableLineaire, correspondant aux trois modalités d'interpolation reconnues par TEMPOS (Cf. § 3.2.1). Ces trois sous-classes ne possèdent pas de méthodes propres, mais elles fournissent des implantations pour certaines méthodes dont elles héritent. En fait, les classes VariableTemporelle et VariableDeValidite sont "virtuelles", au sens où elles ne sont pas destinées à être instanciées car n'implantant pas toutes les méthodes qu'elles définissent. Ainsi, la méthode ObtenirHistorique n'est implantée qu'au niveau de la classe VariableDeTransaction et au niveau des sous-classes de VariableDeValidite.

Ci-dessous, nous donnons la spécification de ces classes dans le langage de définition de schéma d'$O_2$. Cette spécification traduit les interfaces ODMG de la figure 3.12, page 60.

```
class VariableTemporelle /* classe virtuelle */
private type tuple(DomaineObservation : TSequence, HistoriqueEffectif : Historique);
public method obtenirValeur : Object; /* méthode sans implantation */
       method modifierValeur(Object O); /* méthode sans implantation */
       method obtenirHistorique : Historique; /* méthode sans implantation */
       method obtenirHistoriqueEffectif : Historique;
end;
class VariableDeTransaction inherit VariableTemporelle
private type tuple(HistoriqueEffectif : IChronique, /* spécialisation d'un attribut hérité */
                   derniereActivation: Instant, etatActivation : boolean);
public method modifierValeur(Object 0); /* méthode avec implantation */
       method obtenirValeur : Object; /* méthode avec implantation */
       method obtenirHistorique : Historique; /* méthode avec implantation */
       method activer; method desactiver; /* méthode avec implantation */
       method obtenirEtatActivation : boolean; /* méthode avec implantation */
```

```
end;
class VariableDeValidite inherit VariableTemporelle /* classe virtuelle. */
public method modifierValeur(Object V) /* méthode avec implantation */
        method modifierDomaineObservation(TSequence S); /* méthode avec implantation */
        method modifierHistoriqueEffectif(Historique H); /* méthode avec implantation */
end;
class VariableEscalier inherit VariableDeValidite
public method obtenirValeur : Object; /* méthode avec implantation */
        method obtenirHistorique : Historique /* méthode avec implantation */
end; /* Les déclarations des autres classes sont calquées sur cette dernière. */
```

#### Classes temporelles

Les classes temporelles sont codées par deux classes $O_2$ : ObjetDeValidité et ObjetDeTransaction. La spécification de ces deux classes s'obtient par traduction directe des interfaces ODMG de la figure 3.13Γpage 61

#### Associations temporelles

Une association temporelle est codée au travers de deux attributs temporels. Ce codage n'assure pas l'intégrité référentielleΓqui est alors laissée à la charge des programmes d'application. Cette limitation du prototype TEMPOS est héritée des limitations du SGBD $O_2$Γqui n'offre aucun support pour l'implantation de contraintes d'intégrité.

Face à ces limitationsΓnous avons envisagé d'implanter l'intégrité référentielle entre des attributs temporelsΓen nous appuyant sur le gestionnaire de *règles actives* NAOS [Col98]Γ développé au dessus du système $O_2$. Une règle active en NAOSΓest un élément du schéma de la forme "événement-condition-action"Γpermettant de déclencher un traitement lorsqu'un événement se produit. NAOS reconnaît des types d'événements tels que la création d'un objetΓla modification d'un attributΓou l'appel d'une méthode. Ces trois types d'événements suffisent à détecter les opérations pouvant conduire à une violation d'une contrainte d'intégrité référentielleΓque ce soit entre des attributs fugitifs ou temporels. MalheureusementΓle prototype NAOS étant implanté dans une version d'$O_2$ incompatible avec celle sur laquelle est implantée TEMPOSΓnous n'avons pas pu étudier cette piste plus en détails.

## 6.2 Implantation des outils

### 6.2.1 Le préprocesseur TempOQL

Le préprocesseur TEMPOQL a pour rôle de traduire des requêtes en TEMPOQLΓvers des requêtes en OQL faisant intervenir les classes fournies par la bibliothèque "temporelle".

Au cours de cette traduction, le préprocesseur prend en compte les aspects suivants :

- Le mode d'accès de l'application (Cf. § 3.3.3).
- Le typage des valeurs des historiques mis en jeu dans la requête (Cf. § 6.1.2).

### Modes d'accès

Le préprocesseur TempOQL traduit une requête par rapport au mode d'accès de l'application qui en fait la demande. Par défaut, le mode d'accès d'une application est "instantané". En d'autres termes, tout accès à une propriété temporelle retrouve la valeur de son historique à l'instant où la requête est évaluée, et non pas son historique tout entier. Par exemple la requête TempOQL :

```
select e.departement from LesEmployés as e
```

se traduit par défaut (c.à.d. en mode "instantané") en :

```
select ((Departement)e.departement).obtenirValeur
from LesEmployes as e
where e.domaineObservation.contient(instant_present())
/* La fonction instant_present retrouve l'instant associé à l'horloge du système, exprimé à la plus
petite granularité supportée par le système, à savoir, la Seconde. La méthode contient convertit cet
instant à la granularité Jour, et teste si l'instant obtenu figure dans la TSéquence concernée. */
```

À moins que le mode d'accès ne soit explicitement fixé à la valeur "temporel" par l'application, dans quel cas la requête se traduit en :

```
select e.departement.obtenirHistorique from LesEmployes as e
```

La modification du mode d'accès s'effectue en appelant une procédure fournie à cet effet.

### Typage des valeurs des historiques

Comme nous l'avons vu dans le § 6.1.2, le prototype Tempos ne fournit qu'une seule classe pour implanter toutes les instanciations possibles du type paramétré Historique<T>. Cela oblige le développeur à introduire des coercitions explicites chaque fois que la valeur d'un historique à un instant donné est accédée dans un programme ou dans une requête OQL. Grâce au préprocesseur, ces coercitions explicites ne sont pas nécessaires dans le code des requêtes TempOQL. Ainsi, dans la traduction de la requête ci-dessus en mode instantané, on observe que le préprocesseur introduit une coercition dans la clause select.

Pour introduire ces coercitions, le préprocesseur a besoin de connaître le type des propriétés temporelles manipulées. Cette information lui est fournie par le gestionnaire de métadonnées temporelles (voir figure 6.1), qui lui même les obtient du préprocesseur TempODL, lors de la définition du schéma de la base.

Par ailleurs, on rappelle que du fait de la séparation entre types *mutables* et *littéraux* en $O_2$, les instances du type historique au travers d'un type littéral, sont codés par des historiques d'objets mutables, instances d'une classe correspondant à ce type littéral. À titre d'exemple, les historiques dont les valeurs sont censées être de type integer, sont codés par des historiques dont les valeurs sont des objets mutables de la classe suivante :

```
class Integer
public type tuple(literal : integer);
end;
```

Pour rendre transparent ce passage des littéraux aux objets mutables, le préprocesseur traduit une requête de la forme[5] :

```
/* type du résultat (dans le système de types d'O2) : set(integer) */
select e.salaire[@"1/2/98"] from LesEmployes as e
```

en :

```
select ((Integer)e.salaire.obtenirHistorique.Valeur(Instant("1/2/98"))).literal
from LesEmployes as e
```

Bien que cette technique de traduction cache les inconvénients liés au codage du type historique, il ne les élimine pas pour autant. En effet, lors de l'évaluation des requêtes ci-dessus, l'interpréteur OQL doit nécessairement effectuer des vérifications de type. Lorsqu'on manipule un grand nombre d'objets, ces vérifications sont non seulement coûteuses en elles-mêmes, mais en plus, elles sont susceptibles de réduire les possibilités d'optimisation.

### Exemples de traduction

Afin d'illustrer le fonctionnement du préprocesseur, et pour donner une idée des gains en termes d'abstraction et de concision qu'il permet d'obtenir, nous montrons deux exemples de traduction. Le premier fait intervenir une simple restriction selon une condition (requête Q6 page 77). Le second, fait intervenir une version simplifiée de la requête Q3 page 26.

**Q6** *(reprise)* : *Quand l'employé de nom "X" gagne-t-il plus de 2000 ?*

**En TempOQL :**

```
element(select domain(e.salaire as s when s > 5000)
        from LesEmployes as e
        where e.nom = "X")
```

5. Toutes les requêtes que nous exprimons dans la suite, le sont par rapport au mode d'accès "temporel".

**En OQL (généré par le préprocesseur) :**

```
element(select Sequence(select INS.moment /* INS.moment est un intervalle */
                        from select INS
                             from e.salaire.obtenirHistorique.XChronique as INS
                             where ((Integer)INS.valeur).entier > 5000)
        from LesEmployes as e
        where e.nom = "X")
```

Cette traduction est obtenue à partir des deux schémas de traduction suivants.

- C étant une chronique et INS un symbole ne figurant pas dans P :
  C as x when P $\leadsto$ select INS from C as INS where P[x $\leftarrow$ coercition(INS.valeur)]
  Où coercition(E) dénote l'expression obtenue en rajoutant une coercition et éventuellement un déréférencement à l'expression EΓet P[x $\leftarrow$ E] dénote l'expression P dans laquelle toutes les occurrences libres du symbole x sont remplacées par E.
- C étant une chronique et INS un symbole ne figurant pas dans P :
  domain(C) $\leadsto$ Sequence(select INS.moment from C as INS)

**Q21** : *À quels instants, l'unité supervisée par l'employé de nom X a-t-elle eu une production supérieure à 800?*

**En TempOQL :**

```
/* type du résultat : EDI */
element(select domain(superX.supervise.production as p when p > 800)
        from LesSuperviseurs as superX
        where superX.nom = "X")
```

**En OQL (généré par le préprocesseur) :**

```
/* La variable XU dénote un couple ⟨Intervalle, UniteDeProduction⟩ tandis que la variable IP
dénote un couple ⟨Instant, short⟩. */
element(
select Sequence(
      select IP.moment /* bloc select/from traduisant l'opérateur domain */
      from (select IP /* bloc select/from/where traduisant l'opérateur when */
            from (/* bloc select/from/where traduisant la navigation temporelle */
                  select IP
                  from superX.supervise.obtenirHistorique.XChronique as XU,
                     ((UniteDeProduction)XU.valeur).production.obtenirHistorique.IChronique as IP
                  where XU.moment.Contient(IP.moment) ) as IP
                  /* La méthode Contient teste si un instant appartient à un intervalle */
            where ((Integer)IP).literal > 800 ) as IP
from LesSuperviseurs as superX where superX.nom = "X") )
```

On remarque d'après cette traduction que le préprocesseur utilise une représentation par XChronique pour manipuler l'historique des affectations d'un superviseur (puisqu'il s'agit d'un attribut temporel en escalier) alors qu'il utilise une représentation par IChronique pour l'historique de la production d'une unité (s'agissant d'un attribut temporel discret).

#### Choix d'implantation

Le code du préprocesseur a été entièrement écrit dans le langage C à l'aide des outils Lex et Yacc. La communication entre le préprocesseur et le SGBD $O_2$ nécessaire à l'inférence de type s'effectue au travers des fonctions de l'Interface de Programmation (API) du module $O_2$Engine. Cette communication étant encapsulée dans un ensemble de fonctions bien isolées il est envisageable de réutiliser un grande partie du code du préprocesseur dans le contexte d'un autre SGBD qu'$O_2$ à condition bien sûr qu'il implante le langage OQL.

Le préprocesseur est accessible soit en mode embarqué par l'intermédiaire d'une fonction soit en mode interactif grâce à une interface graphique développée à cet effet.

Au cours du développement du préprocesseur nous n'avons pas tenu compte des aspects liées aux performances et à l'optimisation des requêtes générées. En effet le système $O_2$ ne fournit que peu d'indications au sujet des techniques d'optimisation qu'il utilise[6]. Par ailleurs contrairement à des SGBD relationnels tels qu'Oracle[7] il n'est pas possible à notre connaissance d'avoir accès aux plans d'évaluation génerés par l'optimiseur de requêtes. Enfin les techniques d'indexation supportées sont très limitées et il n'est pas possible d'en rajouter de nouvelles.

### 6.2.2 Le moteur de recherche de motifs

Le moteur de recherche de motifs implante l'opérateur sur les historiques Matches décrit dans le § 4.1.3. L'algorithme utilisé pour l'implantation de cet opérateur procède en trois étapes.

D'abord le motif en question est traduit en une expression régulière dans un alphabet composé de variables booléennes. Pendant cette traduction une table de correspondance entre ces variables et les formules atomiques figurant dans la motif est établie. Ainsi le motif :

u.superviseur as sup matches sup.nom = "X" followed by * followed by sup.nom = "Y"

donne lieu à l'expression régulière a.(true)*.b et la table de correspondances :

{ ⟨ a, sup.nom = "X" ⟩, ⟨ b, sup.nom = "Y" ⟩ }.

6. Les références [Clu91, Clu98] décrivent quelques techniques d'optimisation développées dans le cadre du projet ALTAIR ayant débouché sur le SGBD $O_2$. Mais la documentation d'$O_2$ n'explicite pas lequelles de ces techniques sont effectivement utilisées ni dans quelles conditions.

7. http://www.oracle.com

Ensuite, l'expression régulière obtenue est traduite en un automate non-déterministe dont les entrées sont des flots de booléens. Pour effectuer cette étape, nous avons exploité une technique développée dans [Ray96] et matérialisé dans un outil appelé Reglo[8]. Deux raisons justifient le choix de cette technique à base d'automates non-déterministes, par rapport aux techniques classiques de génération d'automates à états finis déterministes [ASU86] :

- Reglo part de l'hypothèse que l'alphabet de l'expression régulière est constitué de variables booléennes, et génère un automate dont les entrées sont des flots de booléens, chacun dénotant la suite de valeurs prises par l'une des variables. Ceci correspond parfaitement à notre besoin. A contrario, dans les techniques de génération d'automates classiques, l'alphabet de l'expression booléenne est composé de lexèmes et l'entrée de l'automate est constituée d'un seul flot de lexèmes. Certes, cette caractéristique ne rend pas les techniques classiques inapplicables à notre contexte, car il est possible de traduire un encodage par variables booléennes en un encodage par lexèmes [Ray96]. Cependant, lors de cette traduction, la taille de l'expression régulière est susceptible d'augmenter considérablement.
- La taille de l'automate non-déterministe généré par Reglo est linéairement proportionnelle à la taille de l'expression régulière mise en jeu, alors que dans les techniques de génération d'automates déterministes classiques, la taille de l'automate est dans le pire des cas exponentielle en la taille de l'expression régulière.

Une fois l'automate généré, il est exécuté en lui fournissant en entrée les résultats des évaluations des formules atomiques du motif, pour chaque instant dans le domaine de l'historique concerné. Ce processus termine lorsque, soit l'automate répond vrai (il a trouvé une occurrence de l'expression régulière), soit le domaine de l'historique est complètement balayé.

Chaque évaluation d'une formule atomique par rapport à un instant, est susceptible d'engendrer un ou plusieurs accès à la base. Toutefois, si la valeur de l'historique ne change pas entre un instant et le suivant, il n'est pas nécessaire de ré-évaluer les formules atomiques pour fournir les entrées correspondant au deuxième instant. Il s'en suit que dans le pire des cas, chaque formule atomique est évaluée autant de fois qu'il y a des changements de valeurs dans l'historique.

L'algorithme ci-dessus possède un inconvénient majeur dès qu'il s'agit de manipuler des motifs comportant des contraintes de durées. En effet, ces motifs donnent lieu à des expressions régulières de la forme[9] $a.b^{[i]}.c$, expression qui est "expansée" en une expression de la forme a.b...b.c (la variable b figurant i fois) avant de générer l'automate correspondant[10].

8. http://www-verimag.imag.fr/~raymond

9. Ici, "." dénote la concaténation et $E^{[i]}$ dénote l'opérateur d'exponentiation des langages réguliers.

10. Cette expansion est effectuée par toutes les techniques de génération d'automates à états finis, qu'ils soient déterministes ou pas.

La taille de l'automate générée dépend alors de la valeur de i et donc de la mesure des durées intervenant dans les contraintes. Ainsi, le motif true during < #"3 jours" évalué sur un historique à la granularité de la minute, donne lieu à un automate d'au moins $3 \times 24 \times 60 = 4320$ états. [Jea98] étudie la possibilité d'étendre Reglo au travers de "compteurs", de façon à éviter cette expansion. Malheureusement, ce travail n'est pas encore assez avancé pour faire l'objet d'une implantation.

### 6.2.3 L'outil de visualisation point-par-point

L'architecture de l'outil de visualisation point-par-point suit un schéma classique dans le domaine de l'Interaction Homme-Machine [Cou90]. Elle est constituée de trois modules : le "noyau fonctionnel", le "contrôleur", et le module de "présentation". Le noyau fonctionnel regroupe les fonctionnalités offertes par le système pour lequel l'interface est conçue (en l'occurrence $O_2$/Tempos). Le module "présentation" gère les objets avec lesquels l'utilisateur interagit (c.à.d. les fenêtres). Le "contrôleur" agit d'intermédiaire entre ces deux modules.

Cette architecture rend le code du module "présentation" réutilisable dans le cadre d'une autre implantation du noyau fonctionnel. Dans notre cas, ceci revient à dire que grâce à cette architecture, le module "présentation" est indépendant du SGBD temporel sous-jacent, à condition que celui-ci fournisse des fonctionnalités équivalentes à celles de TEMPOS.

Plus précisément, le "noyau fonctionnel" de l'outil de visualisation est constitué de la bibliothèque de classes de TEMPOS, augmentée de quelques autres fonctions supplémentaires répondant à des besoins spécifiques à la technique de navigation. Les données manipulées à ce niveau sont des objets (persistants ou pas) gérés par le système $O_2$.

Le "contrôleur" est censé formater les données contenues dans les objets $O_2$ manipulés, en une représentation adaptée à la génération des fenêtres et à la gestion des interactions utilisateur. Cette représentation est obtenue par traduction des diagrammes de classes figurant dans l'annexe C. Le contrôleur implante par ailleurs toutes les méthodes sur ces classes, et en particulier celles décrites dans les diagrammes de collaboration UML de l'annexe C. Certaines de ces méthodes font intervenir des accès à la base de données. Ces accès sont effectués au travers de l'API du module $O_2$Engine.

Dans sa version actuelle, l'implantation du module "présentation" utilise XForms[11] : une boîte à outils graphiques bâtie au dessus du système de fenêtrage X Windows [Nye92]. XForms fournit une grande variété de types d'objets graphiques, comme par exemple, des boutons, des textes, des barres de défilement, des menus, etc. Les objets graphiques sont utilisés pour composer des "formulaires", qui peuvent ensuite être présentés sous diverses géométries et configurations. La figure 6.2 donne une idée des interfaces générées au travers de XForms.

11. http://world.std.com/~xforms

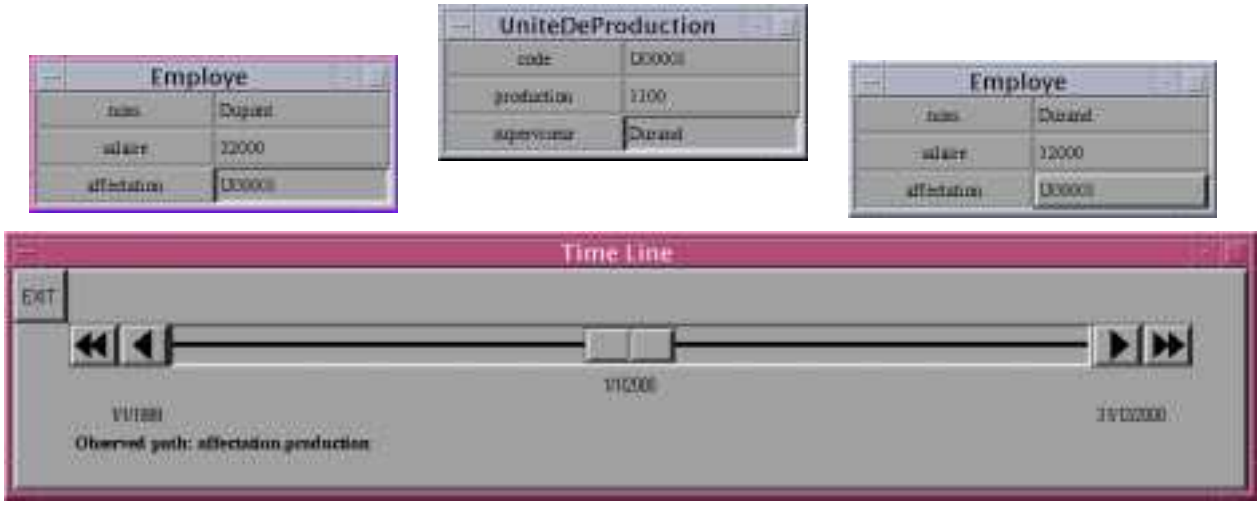



FIG. 6.2 – *Capture d'écran d'une session avec l'outil de visualisation point-par-point.*

Le choix de XForms a été guidé par sa simplicité, son efficacité, et son adéquation vis-à-vis des besoins en matière d'objets graphiques de la technique de visualisation. Grâce à ces caractéristiques, l'implantation du module "présentation" comporte à peine quelques 600 lignes de code C. Toutefois, nous pensons qu'il faut envisager dans le futur une réimplantation de ce module au dessus d'une boîte à outils plus puissante, comme par exemple celle du langage TCL/TK[12] ou la bibliothèque Swing[13] du langage Java. La puissance de ces boîtes devrait permettre l'implantation de certains détails graphiques absents de l'interface actuelle, comme par exemple les liens explicites entre les fenêtres-états (Cf. figure 6.2).

## Récapitulatif

Dans ce chapitre nous avons décrit l'implantation de la plate-forme TEMPOS au dessus du SGBD à objets $O_2$. Cette implantation fait intervenir de nombreux composants de ce système, dont l'API du module $O_2$Engine, le préprocesseur du langage $O_2$C, le générateur d'interfaces $O_2$Look et l'interpréteur OQL. Par ailleurs, elle s'appuie sur divers outils extérieurs à $O_2$ : des générateurs d'analyseurs lexicaux et syntaxiques classiques (Lex et Yacc), une boîte à outils graphiques (XForms), et un générateur/interpréteur d'automates à états finis (Reglo).

Les principaux problèmes que nous avons rencontrés lors de l'implantation, concernent l'absence de classes paramétrées dans le modèle $O_2$. En effet, le type abstrait historique se traduit naturellement en une telle classe. Pour contourner partiellement ce problème, nous avons simulé cette paramétrisation en utilisant les préprocesseurs de TEMPODL et de TEMP-

12. http://www.scriptics.com ou http://www.sco.com
13. http://java.sun.com/products/jfc/tsc

OQL. Toutefois, la solution obtenue n'est pas satisfaisante, et ce à plusieurs égards.

Le manque d'extensibilité et de documentation au niveau des couches basses du SGBD $O_2$ (en particulier au niveau de l'interpréteur OQL) a aussi limité notre expérimentation. En effet, avec plus de connaissance sur ces couches, nous aurions pu mener des études de performances ainsi que des expérimentations de techniques d'optimisation et d'indexation.

Le prototype TEMPOS a servi aussi bien à valider la faisabilité et la cohérence de nos propositions, qu'à mener des expérimentations sur des applications (Cf. chapitre 5). Par ailleurs, il a permis de présenter l'approche TEMPOS au cours de démonstrations publiques, dont [DCFS98] et [DLF+99b].

# Chapitre 7

# Conclusion

– Demain ça dure combien ?
– L'éternité et un jour...

T. Angelopoulos
*L'éternité et un jour*

## 7.1 Bilan

Nous avons présenté un modèle de données, un langage de requêtes et un outil de visualisation, formant une plate-forme (baptisée TEMPOS) pour le développement d'applications temporelles au dessus de SGBD à objets.

Le modèle de données de TEMPOS est une extension conservatrice de l'ODMG, structurée en trois niveaux additifs :

- Un premier niveau constitué d'un ensemble de types modélisant des valeurs temporelles (instants, durées, ensembles d'instants), spécifiés à différents niveaux de granularité par rapport à des repères temporels éventuellement définis par l'utilisateur.
- Un deuxième niveau qui rajoute un type dédié à la notion d'historique.
- Un troisième niveau introduisant les concepts de classe et de propriété temporelles.

Cette structuration permet aux applications de choisir un sous-ensemble du modèle en fonction de leurs besoins. En particulier, le concept de "variable temporelle" (propriété ou classe dont l'évolution au cours du temps est observée) n'est introduit que dans le troisième niveau. C'est donc seulement à ce niveau que sont définis les opérateurs de mise à jour tenant compte du mode d'évolution des objets et de leurs propriétés. Une application qui se contenterait des opérateurs de mise à jour fournis par le modèle d'objets de l'ODMG (à savoir l'affectation d'une valeur à une propriété et la création et destruction d'objets), pourrait alors adopter le deuxième niveau, dans lequel les historiques sont vus exclusivement comme des valeurs "littérales", au même titre que les entiers, les réels, etc. À notre connaissance, cette

séparation entre les historiques vus comme des valeurs, et les propriétés temporelles vues comme des variables, constitue une originalité de notre approche.

Par ailleurs, le troisième niveau du modèle est conçu de façon à permettre la migration transparente d'applications bâties au dessus de bases de données non-temporelles, lorsque les données contenues dans celles-ci sont rendues temporelles par une modification du schéma. Cet aspect a longtemps été négligé dans les recherches sur les modèles de données temporelles, et surtout dans le cadre des approches à objets.

Le langage de requêtes de TEMPOS, à savoir TEMPOQL, se distingue des propositions similaires principalement par trois spécificités :

- Il favorise l'encapsulation des représentations en fournissant des opérateurs de haut niveau permettant de manipuler les historiques sans faire référence à des représentations particulières.
- Il intègre des opérateurs originaux, tels que l'opérateur de recherche de motifs et l'opérateur de regroupement d'un historique selon une durée.
- En appliquant un principe de généralisation d'opérateurs à des listes, largement répandu dans le contexte des langages à flots de données, TEMPOQL étend la plupart des opérateurs d'OQL aux historiques. En particulier, l'extension aux historiques de l'opérateur d'application d'une propriété à un objet (c.à.d. de l'opérateur ".") donne lieu à un opérateur très puissant, qui permet de naviguer au travers des propriétés temporelles au même titre qu'au travers des propriétés non-temporelles.

L'outil de visualisation quant à lui, est fondé sur un paradigme original, qui permet de naviguer de façon orthogonale au travers de la dimension temporelle des objets, au travers des objets composant une collection, et au travers de leurs associations.

La plate-forme TEMPOS a été confrontée à divers types d'applications, dont une concernant le dépouillement d'enquêtes sur les activités et les déplacements d'individus, et une autre concernant la gestion de documents vidéo. Chacune de ces confrontations a mis en évidence des points manquants qui nous ont amenés à affiner les concepts et fonctionnalités de TEMPOS à différents niveaux.

Enfin, dans le but de montrer la faisabilité de l'approche développée, et surtout de servir de support aux expérimentations, le modèle et les interfaces de la plate-forme TEMPOS ont été implantés au dessus du SGBD à objets $O_2$. Cette implantation fait appel à divers outils : des générateurs d'analyseurs lexicaux et syntaxiques, une boîte à outils graphiques, et un générateur/interpréteur d'automates à états finis.

## 7.2 Perspectives

Nous pensons qu'une partie importante des idées présentées dans cette thèse sont assez mûres pour faire l'objet d'un transfert industriel. Bien que certains aspects méritent d'être affinésΓil est notre avis qu'ils relèvent majoritairement d'un travail d'ingénierieΓet non pas d'un projet de recherche à proprement parler.

La prise en compte de l'orthogonalité du temps de validité et du temps de transaction (c.à.d. de la "bi-temporalité" ) au sein de TEMPOSΓconstitue l'une des seules exceptions à cet énoncé. En effetΓde nombreux travaux ([SA85ΓJS94] par exemple) montrent l'intérêt de pouvoir modéliser et manipuler des entités et des associations dont l'évolution est observée simultanément suivant ces deux dimensionsΓalors que TEMPOS ne permet d'observer que l'une d'entre elles à la fois. La séparation entre le deuxième et le troisième niveau du modèle TEMPOS devrait faciliter son extension dans cette direction. En effetΓau deuxième niveauΓla bi-temporalité peut être prise en compte au travers d'historiques d'historiques. Au troisième niveauΓil s'agirait de définir des interfaces correspondant aux concepts d'objet et de propriété bi-temporels. Les méthodes définies sur ces interfaces doiventΓd'une façon ou d'une autreΓregrouper celles définies sur les interfaces "objet de validité"Γ"objet de transaction"Γ"propriété de validité" et "propriété de transaction" que nous avons décrites dans la section 3.2.

Ce point mis à partΓnos propositions de poursuites à cette thèse portent essentiellement sur des besoins dégagés au cours de la conception de TEMPOSΓmais dont la portée dépasse son spectre d'applications.

### Indexation d'objets mobiles sur des réseaux contraints

Nous partageons avec beaucoup d'autres chercheurs et d'industrielsΓl'idée que la représentation et l'indexation d'objets mobiles est un domaine très porteurΓriche en défis et en subtilités dont une bonne compréhension passe par des recherches approfondies.

En effetΓde nombreuses technologies en plein essor telles que la téléphonie mobileΓles transports à la demandeΓet le suivi de la diffusion de substances polluantesΓfont intervenir des modules logiciels manipulant des grandes quantités de données relatives à des positions et des trajectoires d'objets se déplaçant dans un espace tri-dimensionnel (deux dimensions spatiales et une temporelle).

Ces données sont généralement exploitées soit en temps réel dans des processus de prise de décision automatisés (tel taxi doit desservir tel client)Γsoit de façon différéeΓdans des analyses statistiques permettant de dégager des tendances futuresΓet éventuellement de prendre des décisions en conséquence (un nouveau terminus de taxis est nécessaire dans telle zone).

Pour faire face aux défis posés par ces applicationsΓde nombreux travaux de recherche dans le domaine des bases de données ont été entamés ces dernières années. Ces travaux ont

donné lieu à des modèles de données et des langages de requêtes pour bases d'objets mobiles [WXCJ98, EGMS99], mais aussi à des techniques d'indexation en mémoire secondaire, permettant soit de suivre des collections d'objets mobiles dont les positions et les vecteurs de vitesse arrivent en temps réel [SJLL00], soit de représenter et de manipuler des collections de trajectoires "passées" [PTJ00].

La plupart des techniques d'indexation issues de ces travaux font l'hypothèse que les objets se déplacent librement dans l'espace. Or, ceci est loin d'être le cas dans des applications de type "transports à la demande" où les mouvements des objets sont contraints par un réseau routier. Bien que ce facteur ne rende pas inexploitables les techniques s'appliquant à des objets aux trajectoires libres, nous pensons que la prise en compte des contraintes imposées par le réseau sur lequel se déplacent les objets peut déboucher sur des techniques plus efficaces, voire plus simples, que celles développées dans les travaux référencés ci-dessus, au prix éventuellement d'une perte de généralité.

### Optimisation de requêtes temporelles

Les requêtes OQL générées par le préprocesseur TempOQL comportent généralement de nombreux emboîtements d'itérateurs (Cf. § 6.2.1). Autrement dit, il s'agit de requêtes de la forme :

```
select ...
from (select ...
        from (select ... from ...)
        where ...)
where exists(select ... from ... where ...)
```

Une manière naïve d'évaluer ce type de requêtes, consiste à commencer par les itérateurs les plus emboîtés, et à remonter progressivement vers des itérateurs moins emboîtés. Toutefois, ce mode d'évaluation conduit en général à des mauvaises performances.

Aussi, des techniques d'optimisation ont été développées, qui permettent de réduire le nombre de niveaux d'emboîtements dans une requête [CM93, Feg98]. Ces techniques partent de l'idée qu'une expression comportant un itérateur emboîté dans un autre peut, dans certains cas, être traduite en une expression équivalente où ces itérateurs sont fusionnés en un seul. L'itérateur résultant, qui comporte nécessairement plusieurs collections dans la clause from, peut ensuite être évalué au travers d'algorithmes classiques de jointure relationnelle [ME92].

Une perspective intéressante consiste à examiner dans quelle mesure ces techniques d'optimisation s'appliquent aux requêtes générées par TempOQL. Si cela n'était pas le cas, il s'agirait alors de concevoir des nouvelles techniques (en s'appuyant sur le degré d'abstraction

fourni par les opérateurs de TempOQL) et de les intégrer soit au niveau du préprocesseur soit directement au niveau d'un optimiseur d'OQL.

Par ailleurs, lorsque les historiques sont représentés au travers de XChroniques (Cf. § 6.1.1), les opérateurs de mise en correspondance d'historiques (jointures temporelles et navigation point-par-point) se traduisent en des jointures de XChroniques faisant intervenir des conditions d'intersection entre intervalles. Or, de nombreux algorithmes pour ce type de jointures ont été proposés [Seg93, SSJ94, Zur97]. L'intégration de ces algorithmes dans un interpréteur d'OQL et l'évaluation des gains en performance qu'ils procurent constitue une autre poursuite possible de notre travail.

### Visualisation de données temporelles

L'étude autour de la visualisation de bases d'objets temporels mérite d'être poursuivie dans au moins deux directions :

- **Exploration de grandes quantités d'information.** La majorité des techniques de visualisation d'objets décrites dans la section 4.2 s'avèrent inadaptées lorsqu'elles opèrent sur des grandes quantités d'information. Or, la manipulation de volumes importants de données est justement l'une des problématiques auxquelles les SGBD sont confrontés.
- **Prise en compte de types de données spécifiques.** Nos contacts avec des utilisateurs et des applications nous ont permis de dégager le besoin de techniques de visualisation d'objets temporels prenant en compte des aspects tels que la "multigranularité", la "périodicité" et plus généralement la "cyclicité". En ce sens, le travail démarré dans [Daa99, DDF+00] nous semble très prometteur. L'étape suivante consiste à comparer et à intégrer ce travail avec les nombreuses techniques de visualisation utilisées dans le domaine du traitement statistique des données.

  Par ailleurs, l'émergence de nombreuses applications manipulant des objets mobiles (objets comportant une dimension spatiale dont l'évolution au cours du temps est observée) soulève de nouveaux besoins en matière d'outils de visualisation, vis-à-vis desquels il semble pertinent de considérer des techniques d'animation d'interfaces [Vod97].

De façon plus immédiate, il serait bien sûr intéressant d'évaluer la technique de visualisation temporelle point-par-point que nous avons proposée, par une étude auprès d'utilisateurs potentiels.

# Références électroniques

[1] Computer Associates: Jasmine. http://www.cai.com/products/jasmine.htm.

[2] CROQUE. http://www.fmi.uni-konstanz.de/~gluche/croque.

[3] Infobject: ObjectDRIVER. http://www.infobjects.net.

[4] lambda-DB. http://lambda.uta.edu/lambda-DB.

[5] LOTUS 1-2-3. http://www.lotus.com/home.nsf/tabs/lotus123.

[6] Microsoft ACCESS. http://www.microsoft.com/office/access.

[7] Objectivity. http://www.objectivity.com

[8] Oracle. http://www.oracle.com.

[9] OSF Motif. http://www.motifzone.com.

[10] Poet Software: Object Server Suite. http://www.poet.com.

[11] Reglo. http://www-verimag.imag.fr/~raymond.

[12] Sun Microsystems: JavaBlend. http://www.sun.com/software/javablend.

[13] Sun Microsystems: Java Swing. http://java.sun.com/products/jfc/tsc.

[14] TCL/TK. http://www.scriptics.com.

[15] TIGER. http://www.cs.auc.dk/~boehlen.

[16] TimeConsult: TimeDB. http://www.timeconsult.com.

[17] TOOBIS ESPRIT Project. http://www.mm.di.uoa.gr/~toobis.

[18] Versant Inc. http://www.versant.com.

[19] XForms. http://world.std.com/~xforms.

# Bibliographie

[ACC+96] S. AdaliΓK. Selçuk CandanΓS.S. ChenΓK. Erol et V.S. Subrahmanian. The Advanced Video Information System: data structures and query processing. *Multimedia Systems*Γ4:172 – 186Γ1996.

[Adi96] M. Adiba. STORM: An object-orientedΓmultimedia DBMS. Dans K. Nwosu et al.Γéditeur Γ*Multimedia Database Management Systems. Design and Implementation Strategies*Γ chapitre 3Γ pages 47–88. Kluwer Academic PublishersΓ 1996.

[AHP97] E. AndonoffΓG. Hubert et A. Le Parc. Interrogation de bases de données intégrant des versions. Dans *Actes du XV congrès INFORSID*ΓToulouseΓFranceΓ Juin 1997.

[AJ97] T. Abdessalem et G. Jomier. VQL : un langage de requêtes pour bases de données multiversions. Dans *Actes des 13èmes Journées Bases de Données Avancées (BDA)*ΓGrenobleΓFranceΓSeptembre 1997.

[Ala97] S. Alagic. The ODMG model: does it makes sense? Dans *Proc. of the Int. Conference on Object-Oriented Programming Systems, Langages and Applications (OOPSLA)*ΓAtlantaΓGA (USA)ΓOctober 1997.

[Ala99] S. Alagic. A family of the ODMG object models. Dans *Proc. of the east-european conference on Advanced Databases and Information Systems (ADBIS)*ΓMariborΓ SloveniaΓSeptember 1999.

[All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*Γ26(11)ΓNovember 1983.

[ASU86] A. AhoΓR. Sethi et J. Ullman. *Compilers : principles, techniques and tools.* Addison-WesleyΓ1986.

[BBJS97] J. BairΓM. BöhlenΓC.S. Jensen et R.T. Snodgrass. Notions of upward compatibility of temporal query languages. Rapport de recherche TR-6ΓTime CenterΓ 1997.

[BDK92] F. BancilhonΓC. Delobel et P. KanellakisΓéditeurs. *The story of $O_2$*. Morgan KaufmannΓ1992.

[Ber81] J. Bertin. *Graphics and Graphic Information Processing*. Walter de Gruyter & CoΓBerlinΓ1981.

[BF97] B. Benatallah et M.-C. Fauvet. Le point sur l'évolution du schéma d'une base d'objets. *L'Objet : logiciels, bases de données, réseaux*Γ3(3)Γ1997.

[BF99] B. Benatallah et M.-C. Fauvet. Chapitre VIIIΓEvolution de schéma. Dans M. OussalahΓéditeurΓ*Génie Objet*. Hermès Science PublicationsΓJuin 1999.

[BFG96] E. BertinoΓE. Ferrari et G. Guerrini. A formal temporal object-oriented data model. Dans *Proc. of the 5th Int. Conference on Extending DataBase Technology (EDBT)*ΓAvignonΓFranceΓMarch 1996.

[BFGM98] E. BertinoΓE. FerrariΓG. Guerrini et I. Merlo. Extending the ODMG object model with time. Dans *Proc. of the European Conference on Object-Oriented Programming (ECOOP)*ΓBrusselsΓBelgiumΓJuly 1998.

[Böh95] M. Böhlen. Temporal database system implementations. *SIGMOD record*Γ 24(4):53–60ΓDecember 1995.

[Can97] J.-F. Canavaggio. TEMPOSΓun modèle d'historiques pour un SGBD temporel. Thèse de doctoratΓUniversité Joseph FourierΓGrenobleΓNovembre 1997.

[Cat94] R.G.G. Cattell et al.Γéditeur. *The Object Database Standard : ODMG-93*. Morgan KaufmannΓ1994.

[CB97] R.G.G. Cattell et D. BarryΓéditeurs. *The Object Database Standard: ODMG 2.0*. Morgan KaufmannΓ1997.

[CB00] R.G.G. Cattell et D. BarryΓéditeurs. *The Object Database Standard: ODMG 3.0*. Morgan KaufmannΓJanuary 2000.

[CC93] J. Clifford et A. Croker. The Historical Relational Data Model (HRDM) revisited. In Tansel et al. [TCG+93].

[CC97] J. Celko et J. Celko. Debunking object database myths. *Byte Magazine*ΓOctober 1997. http://www.byte.com/art/9710/sec6/art6.htm.

[CCLB97] T. CatarciΓM. CostabileΓS. Levialdi et C. Batini. Visual query systems for databases: a survey. *Journal of visual languages and computing*Γ8(2)Γ1997.

[CCT94] J. CliffordΓA. Croker et A. Tuzhilin. On completeness of historical relational query languages. *ACM Transactions on Database Systems*Γ19(1):64 – 116Γ1994.

[CD97] J.-F. Canavaggio et M. Dumas. Manipulation de valeurs temporelles dans un SGBD à objets. Dans *Actes du 15e congrès INFORSID*ΓToulouseΓjuin 1997.

[CDI+97] J. CliffordΓC. DyresonΓT. IsakowitzΓC. Jensen et R. Snodgrass. On the semantics of "Now" in databases. *ACM Transactions on Database Systems*Γ22(2):171 – 214ΓJune 1997.

[CEF98] A. CarlsonΓS. Estepp et M. Fowler. Temporal patterns. Dans *Proc. of the Pattern Languages of Programs Conference (PLoP)*ΓMonticelloΓIL (USA)Γ1998. Proceedings published as technical report WUCS-98-25ΓUniversity of Washington.

[Cel95] J. Celko. *SQL for smarties : advanced SQL programming.* Morgan KaufmannΓ 1995.

[CG94] T. Cheng et S. Gadia. A pattern matching language for spatio-temporal databases. Dans *Proc. of the 3rd Int. Conference on Information and Knowledge Management (CIKM)*ΓGaithersburgΓMD (USA)ΓNovember 1994.

[Cha99] S. Chardonnel. Emplois du temps et de l'espace - Pratique des populations d'une station touristique de montagne. Thèse de doctoratΓUniversité Joseph FourierΓ Grenoble (France)Γjanvier 1999.

[CHMW96] M. CareyΓ L. HaasΓ V. Maganty et J. Williams. PESTO : an integrated query/browser for object databases. Dans *Proc. of the Int. Conference on Very Large Databases (VLDB)*ΓMumbaiΓIndiaΓ1996.

[Cho94] J. Chomicki. Temporal query languages: a survey. Dans *Proc. of the International Conference on Temporal Logic*ΓBonnΓDEΓ1994.

[CK86] H-T. Chou et W. Kim. A unifying framework for version control in a CAD environment. Dans *Proc. of the 12th Int. Conference on Very Large DataBases (VLDB)*ΓKyotoΓJapanΓAugust 1986.

[CLMS94] J.P. CheylanΓS. LardonΓH. Mathian et L. Sanders. Les problématiques temporelles dans les SIG. *Revue Internationale de géomatique*Γ4(3-4)Γ1994.

[Clu91] S. Cluet. Langages et optimisation de requêtes pour systèmes de gestion de bases de données orientés-objet. Thèse de doctoratΓUniversité de Paris-SudΓ Juin 1991.

[Clu98] S. Cluet. Designing OQL: allowing objects to be queried. *Information Systems*Γ 23(5):279 – 305Γ1998.

[CM93] S. Cluet et G. Moerkotte. Nested queries in object bases. Dans *Proc. of the Int. Workshop on Database Programming Languages (DBPL)*ΓNew YorkΓNY (USA)Γ1993.

[CM95] Guy Cousineau et Michel Mauny. *Approche fonctionnelle de la programmation.* EdiscienceΓParisΓ1995.

[Cod70] E. F. Codd. A relationnal model of data for large shared data banks. *Communications of ACM*Γ13(6)ΓJune 1970.

[Col98] C. Collet. NAOS. Dans N. PatonΓéditeurΓ*Active Rules in Databases.* Springer VerlagΓ1998.

[Cou90] J. Coutaz. *Interfaces homme-ordinateur: conception et realisation.* Dunod informatiqueΓ1990.

[CPHP87] P. CaspiΓD. PilaudΓN. Halbwachs et J. Plaice. LUSTRE : a declarative language for programming synchronous systems. Dans *Proc. of the 14th ACM Symposium on Principles of Programming Languages*ΓMunchenΓGermanyΓ1987.

[CR87] J. Clifford et A. Rao. A simple general structure for temporal domains. In Rolland et al. [RBL87].

[CSF96] J.-F. CanavaggioΓP.-C. Scholl et M.-C. Fauvet. Un modèle multigranulaire du temps pour un SGBD temporel. Rapport de recherche RR 962-IΓLaboratoire LSR-IMAGΓoctobre 1996.

[CT95] J. Clifford et A. TuzhilinΓéditeurs. *Proc. of the workshop on Recent Advances in Temporal Databases*ΓWorkshops in ComputingΓZurichΓSwitzerlandΓSeptember 1995. Springer Verlag.

[CZ99] C.X. Chen et C. Zaniolo. Universal temporal extensions for data languages. Dans *Proc. of the 15th Int. Conference on Data Engineering (ICDE)*ΓSydneyΓ AustraliaΓ1999.

[DA83] C. Delobel et M. Adiba. *Bases de données et systèmes relationnels.* Dunod InformatiqueΓ1983.

[DAA+95] Y. DennebouyΓM. AnderssonΓA. AuddinoΓY. DupontΓE. FontanaΓM. Gentile et S. Spaccapietra. SUPER: visual interfaces for object + relationship data models. *Journal of visual languages and computing*Γ6(1):27 – 52Γ1995.

[Daa99] C. Daassi. Techniques de visualisation de données temporelles. Rapport de rechercheΓDEA Systèmes d'InformationΓUniversités de Grenoble et Université de SavoieΓjuin 1999.

[Dar98] H. Darwen. Valid time and transaction-time proposals: language design aspects. In Etzion et al. [EJS98].

[DC98] J. Dionisio et A. Cárdenas. A unified data model for representing multimediaΓtime-line and simulation data. *IEEE Transactions on Knowledge and Data Engineering*Γ10(5)ΓOctober 1998.

[DCFS98] M. DumasΓJ.-F. CanavaggioΓM.-C. Fauvet et P.-C. Scholl. TEMPOS: managing time and histories on top of OO DBMS. Dans *Proc. of Int. Conference on Extending DataBase Technology (prototype demonstrations session)*Γ ValenciaΓ SpainΓMarch 1998.

[DDF+00] C. DaassiΓM. DumasΓM.C. FauvetΓL. Nigay et P.C. Scholl. Interactively exploring temporal object databases. Dans *Actes des 16èmes Journées Bases de Données Avancées (BDA)*ΓBloisΓOctobre 2000. À paraître.

[DDFN00] M. DumasΓC. DaassiΓM.C. Fauvet et L. Nigay. Pointwise temporal object database browsing. Dans *Proc. of the ECOOP Symposium on Objects and Databases*Γ Sophia AntipolisΓFranceΓJune 2000.

[DFS98] M. DumasΓ M.-C. Fauvet et P.-C. Scholl. Handling temporal grouping and pattern-matching queries in a temporal object model. Dans *Proc. of the Int. Conference on Information and Knowledge Management (CIKM)*ΓBethesdaΓMD (USA)ΓNovember 1998. ACM Press.

[DFS99a] M. DumasΓM.-C. Fauvet et P.-C. Scholl. TEMPOS: A Temporal Database Model Seamlessly Extending ODMG. Research Report 1013-I-LSR-7ΓLSR-IMAGΓ Grenoble (France)ΓMarch 1999.

[DFS99b] M. DumasΓ M.-C. Fauvet et P.-C. Scholl. Updates and application migration support in an ODMG temporal extension. Dans *Proc. of the Int. Workshop on Evolution and Change in Data Management*ΓParisΓFranceΓNovember 1999.

[DFS00] M. DumasΓ M.-C. Fauvet et P.-C. Scholl. Modèles et langages pour données temporelles. Dans G. Jomier et A. DoucetΓéditeursΓ*Bases de Données*. HermèsΓ ParisΓ2000. À paraître.

[DGJS95] S. Dar, N.H. Gehani, H.V. Jagadish et J. Srinivasan. Queries in an object-oriented graphical interface. *Journal of visual languages and computing*, 6(1):27 – 52, 1995.

[DLF+99a] M. Dumas, R. Lozano, M.-C. Fauvet, H. Martin et P.-C. Scholl. A sequence-based object-oriented model for video databases. Research Report RR 1024-I-LSR 12, LSR-IMAG, Grenoble (France), November 1999.

[DLF+99b] M. Dumas, R. Lozano, A. Front-Conte, G. Vargas Solar et H. Grazziotin-Ribeiro. STORM : serveur réactif d'objets temporels multimédias. Dans *Actes du XVII congrès INFORSID (démonstration de prototype)*, Juin 1999.

[DLF+00] M. Dumas, R. Lozano, M.-C. Fauvet, H. Martin et P.-C. Scholl. Orthogonally modeling video structuration and annotations. Dans *Proc. of the AAAI Workshop on Spatial and Temporal Granularities*, Austin, TX (USA), July 2000. AAAI Press.

[DS95] C.E. Dyreson et R. T. Snodgrass. A timestamp representation. In Snodgrass [Sno95b], chapitre 25.

[EC94] J. Estublier et R. Casallas. The Adele configuration manager. Dans W. Tichy, éditeur, *Configuration Management*, pages 99–139. J. Wiley and Sons, 1994.

[EGMS99] M. Erwig, R.H. Güting et M. Vazirgiannis M. Schneider. Spatio-Temporal Data Types: an approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):269–296, September 1999.

[EJA+97] A.K. Elmagarmid, H. Jiang, A.H. Abdelsalam, A. Joshi et M. Ahmed. *Video Database Systems: Issues, Products and Applications.* Kluwer Academic Publisher, 1997.

[EJS98] O. Etzion, S. Jajodia et S.M. Sripada, éditeurs. *Temporal Databases: Research and Practice.* Springer Verlag, LNCS 1399, 1998.

[Eme90] E.A. Emerson. Temporal and modal logic. Dans J. van Leeuwen, éditeur, *Handbook of Theoretical Computer Science*, volume 2. Elsevier, 1990.

[EPP93] N. Edelweiss, J. Palazzo et B. Pernici. An object-oriented temporal model. Dans *Proc. of the Conference on Computer-Aided Information Software Engineering (CAISE)*, Paris, France, June 1993. Springer Verlag.

[FCD+98] M.-C. FauvetΓS. ChardonnelΓM. DumasΓP.-C. Scholl et P. Dumolard. Analyse de données géographiques : application des bases de données temporelles. *Revue Internationale de Géomatique*Γ8(1-2)ΓNovembre 1998.

[FCD+99] M.-C. FauvetΓS. ChardonnelΓM. DumasΓP.-C. Scholl et P. Dumolard. Applying temporal databases to geographical data analysis. Dans *Proc. of the DEXA Workshop on spatio-temporal data models and languages*ΓFlorenceΓItalyΓ September 1999. IEEE Computer Society.

[FCS97] M.-C. FauvetΓJ.-F. Canavaggio et P.-C. Scholl. Modelling histories in object DBMS. Dans *Proc. of the 8th Int. Conference on Database and Expert Systems Applications (DEXA)*ΓToulouse (France)ΓSeptember 1997. Springer Verlag.

[FDS99] M.-C. FauvetΓM. Dumas et P.-C. Scholl. A representation independent temporal extension of ODMG's Object Query Language. Dans *actes des 15èmes Journées Bases de Données Avancées (BDA)*ΓBordeauxΓFranceΓOctobre 1999.

[FE98] L. Fegaras et R. Elmasri. A temporal object query language. Dans *Proc. of the Int. Workshop on Temporal Representation and Reasoning (TIME)*ΓSanibel IslandΓFL (USA)ΓMay 1998.

[Feg98] L. Fegaras. Query unnesting in object-oriented databases. Dans *Proc. of the ACM SIGMOD Int. Conference on Management of Data*ΓSeattleΓWA (USA)Γ June 1998.

[Feg99] L. Fegaras. VOODOO : a visual object-oriented database language for ODMG OQL. Dans *Proc. of the ECOOP Workshop on Object-Oriented Databases*ΓLisbonΓPortugalΓJune 1999.

[FGNS00] L. ForlizziΓR.H. GütingΓE. Nardelli et M. Schneider. A data model and data structures for moving objects databases. Dans *Proc. of the ACM SIGMOD Conference on Management of Data*ΓDallasΓTX (USA)ΓMay 2000.

[FM95] L. Fegaras et D. Maier. Towards an effective calculus for object query languages. Dans *Proc. of the ACM SIGMOD Int. Conference on Management of Data*ΓSan JoseΓCA (USA)ΓMay 1995.

[FRM94] C. FaloutsosΓM. Ranganathan et Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. Dans *Proc. of the ACM SIGMOD Int. Conference on Management of Data*Γpages 419 – 429Γ1994.

[FS97] M. Fowler et K. Scott. *UML Distilled: Applying the Standard Object Modeling Language*. Addison WesleyΓ1997.

[FSC97] S. FernandesΓU. Schiel et T. Catarci. Visual query operators for temporal databases. Dans *Proc. of the 4th International Workshop on Temporal Representation and Reasoning (TIME)*Γ1997.

[GBE+98] R. GütingΓ M. BöhlenΓ M. ErwigΓ C. JensenΓ N. LorentzosΓ M. Schneider et M. Vazirgianis. A foundation for representing and querying moving objects. Rapport de recherche 238ΓFerUniversität das Hagen (Germany)Γ1998.

[GJ94] S. Gançarski et G. Jomier. Gestion de versions d'entités dans leur contexte : une approche formelle. Dans *actes des 10èmes Journées Bases de Données Avancées (BDA)*ΓClermont FerrandΓFranceΓAoût 1994.

[GJ99] H. Gregersen et C.S. Jensen. Temporal entity-relationship models – A survey. *Transactions on Knowledge and Data Engineering (TKDE)*Γ11(3):464–497Γ 1999.

[GLOS97] I. A. GoralwallaΓY. LeontievΓT. Ozsu et D. Szafron. Modeling temporal primitives : back to basics. Dans *Proc. of Int. Conference on Information and Knowledge Management (CIKM)*ΓLas VegasΓNE (USA)ΓNovember 1997.

[GN93] S. K. Gadia et S. S. Nair. Temporal databases: a prelude to parametric data. In Tansel et al. [TCG+93].

[GO93] I. A. Goralwalla et M.T. Ozsu. Temporal extensions to a uniform behavioral object model. Dans *Proc. of the 12th International Conference on the Entity-Relationship Approach (ER)*ΓArlingtonΓTX (USA)Γ1993.

[GOS98] I. GoralwallaΓM.T. Ozsu et D. Szafron. An object-oriented framework for temporal data models. In Etzion et al. [EJS98].

[GR94] M. Gandhi et E. Robertson. A data model for audio-video data. Dans *Proc. of the 6th Int. Conference on Management of Data (CISMOD)*ΓBangalore (India)Γ 1994. McGraw-Hill.

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*Γ25(2):73–170Γ1993.

[GRS98] S. GrumbachΓP. Rigaux et L. Segoufin. Spatio-temporal data handling with constraints. Dans *Proc. of the ACM Int. Symposium on Geographic Information Systems (ACM-GIS)*ΓBethesdaΓMD (USA)ΓNovember 1998. ACM Press.

[GRS99] S. GrumbachΓP. Rigaux et L. Segoufin. Modeling and querying interpolated spatial data. Dans *actes des 15èmes Journées Bases de Données Avancées (BDA)*Γ BordeauxΓFranceΓOctobre 1999.

[Gut84] A. Guttman. R-trees : A dynamic structure for spatial searching. Dans *Proc. of the ACM SIGMOD Conference on Management of Data*Γpages 47 – 57Γ1984.

[HMS95] R. HjelsvoldΓR. Midtstraum et O. Sandsta. A temporal foundation of video databases. In Clifford and Tuzhilin [CT95].

[HS95] E. Hwang et V.S. Subrahmanian. Querying video libraries. Rapport de rechercheΓ Univ. of MarylandΓ1995.

[Inf97] Informix. *Informix Time Series DataBlade Module*. Informix Software Inc.Γ1997.

[Ins92] American National Standards InstituteΓéditeur. *American National Standard for Information Systems - Database Language - SQL. ANSI X3, 135-1992. Revision and consolidation of ANSI X3, 135-1989 and ANSI X3, 168-1989*. 1992.

[JD98] C.S. Jensen et C.E. Dyreson (Eds). The consensus glossary of temporal database concepts – February 1998 version. In Etzion et al. [EJS98].

[Jea98] B. Jeannet. Expressions régulières avec compteurs. ManuscritΓ1998.

[JS94] C.-S. Jensen et R. Snodgrass. Temporal specialization and generalization. *IEEE Transactions on Knowledge and Data engineering*Γ 6(6):954 – 974Γ December 1994.

[JS99a] K.J. Jacob et D. Shasha. FinTime: a financial time series benchmark. http://cs.nyu.edu/cs/faculty/shasha/fintime.htmlΓ1999.

[JS99b] C.S. Jensen et R.T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*Γ11(1)Γ1999.

[KG95] V. Kouramajian et M. Gertz. A visual query editor for temporal databases. Dans M.P. PapazoglouΓéditeurΓ*Proc. of the 14th Int. Conference on Object-Oriented & Entity Relationship Modelling*ΓGold CoastΓAustraliaΓDecember 1995.

[KT96] I. Kakoudakis et B. Theodoulidis. The TAU Temporal Object Model. Rapport de recherche TR-96-4ΓTimeLabΓUniversity of Manchester (UMIST)Γ1996.

[Lan98] M. Lange. Time patterns. Dans *3rd European Conference on Pattern Languages of Programming and Computing*ΓKloster IrseeΓGermanyΓ1998.

[LEW96] J. Y. LeeΓR. Elmasri et J. Won. Specification of calendars and time series for temporal databases. Dans *Proc. of International Conference on Conceptual Modeling (ER)*ΓCottbusΓGermanyΓOctober 1996.

[LGOS97] J. LiΓI. GoralwallaΓM.T. Ozsu et D. Szafron. Modeling video temporal relationships in an object database management system. Dans *Proc. of IS&T/SPIE Int. Symposium on Electronic Imaging: Multimedia Computing and Networking*Γ pages 80 – 91ΓSan JoseΓCA (USA)Γ1997.

[LM98] R. Lozano et H. Martin. Querying virtual videos with path and temporal expressions. Dans *proc. of the ACM Symposium on Applied Computing*ΓAtlantaΓ Georgia (USA)ΓFebruary 1998.

[LMF86] B. LebanΓD. McDonald et D. Forster. A representation for collections of temporal intervals. Dans *Proc. of the 5th National Conference on Artificial Intelligence (AAAI)*ΓPhiladelphiaΓPA (USA)Γ1986. AAAI press.

[LMR96] R. Laurini et F. Milleret-Raffort. *Les bases de données en géomatique.* HermèsΓ ParisΓ1996.

[Lor93] N. A. Lorentzos. The Interval-eXtended Relational Model and its application to valid-time databases. In Tansel et al. [TCG+93].

[Loz00] R. Lozano. Intégration de données vidéo dans un sgbd à objets. Thèse de doctoratΓUniversité Joseph FourierΓGrenobleΓAvril 2000.

[MBFG99] I. MerloΓE. BertinoΓE. Ferrari et G. Guerrini. A temporal object-oriented data model with multiple granularities. Dans *Proc. of the 6th Int. Workshop on Temporal Representatation and Reasoning (TIME)*ΓOrlandoΓFL (USA)ΓMay 1999.

[ME92] P. Mishra et M. Eich. Join processing in relational databases. *ACM Computing Surveys*Γ24(1)ΓMarch 1992.

[MS91] E. MacKenzie et R. Snodgrass. Evaluation of relational algebra incorporating the time dimension in databases. *ACM Computing Surveys*Γ23(4)Γ1991.

[NL99] F. Nack et A. T. Lindsay. Everything You Wanted to Know About MPEG-7. *IEEE MultiMedia Magazine*Γ6(3–4):65–77ΓJuly 1999.

[Nye92] A. Nye. *Xlib programming manual – 3rd edition.* O'ReillyΓ1992.

[NYK97] K. NakayamaΓK. Yamaguchi et S. Kawai. I-Regular Expression : Regular Expression with Continuous Interval Constraints. Dans *Proc. of the 6th Int. Conference on Information and Knowledge Management (CIKM)*Γpages 40 – 50ΓLas VegasΓ USAΓNovember 1997.

[OPS+95] T. OzsuΓR. PetersΓD. SzafronΓB. IraninΓA. Lipka et A. Muñoz. TIGUKAT: A uniform behavioral objectbase management system. *The VLDB Journal*Γ 4(3):445–492Γ1995.

[Ora97] Oracle. *Time Series Cartridge User's Guide*. Oracle CorporationΓ1997.

[OT93] E. Oomoto et K. Tanaka. OVID : Design and implementation of a video-object database system. *IEEE Transactions on Knowledge and Data Engineering*Γ5(4)Γ August 1993.

[Pat98] N. PatonΓéditeur. *Active Rules in Databases*. Springer VerlagΓ1998.

[PBL+92] D. PlateauΓP. BorrasΓD. LevequeΓJ.C. Mamou et D. Tallot. Building user interfaces with Looks. In Bancilhon et al. [BDK92].

[PMR+96] C. PlaisantΓB. MilashΓA. RoseΓS. Widoff et B. Schneiderman. LifeLines : Visualizing Personal Histories. Dans *Proc. of the ACM CHI Conference on Human Factors in Computing*ΓVancouverΓCanadaΓApril 1996.

[PQ96] D. Peuquet et L. Qian. An integrated database design for temporal GIS. Dans *7th international symposium on Spatial Data Handling*ΓAugust 1996.

[PSZ+97] C. ParentΓS. SpaccapietraΓE. ZimanyiΓP. DoniniΓC. PlazanetΓC. VangenotΓ N. RognonΓJ. Pouliot et P.-A. Crausaz. MADSΓmodèle conceptuel pour des applications spatio-temporelles. *Revue Internationale de géomatique*Γ7Γ1997.

[PTJ00] D. PfoserΓY. Theodoridis et C.S. Jensen. Novel approaches in query processing for moving objects. Dans *Proc. of the Int. Conference on Very Large DataBases (VLDB)*ΓCairoΓEgyptΓAugust 2000.

[Ray96] P. Raymond. Recognizing regular expressions by means of dataflows networks. Dans *Proc. of the Int. Colloquium on Automata, Languages, and Programming, (ICALP)*ΓPaderbornΓGermanyΓJuly 1996. Springer Verlag.

[RBL87] C. RollandΓF. Bodart et M. LéonardΓéditeurs. *Proc. of the IFIP Working Conference on Temporal Aspects in Information Systems*ΓSophia-AntipolisΓMay 1987. AFCET.

[RGMS99] J.F. RoddickΓF. GrandiΓF. Mandreoli et M.R. Scalas. Towards a model for spatio-temporal schema selection. Dans *Proc. of the DEXA Workshop on spatio-temporal data models and languages*Γ FlorenceΓ ItalyΓ September 1999. IEEE Computer Society.

[RKLL98] E. RyuΓY. KimΓM.Y. Lee et K. Lee. Structured modeling for video databases. Dans *proc. of the ER '98 Workshop on Collaborative Work Support and Spatio-Temporal Data Management*. Springer VerlagΓNovember 1998.

[RS91] E. Rose et A. Segev. TOODM – A Temporal Object-Oriented Data Model with temporal constraints. Dans *Proc. of the Int. Conference on the Entity Relationship Approach*ΓSan MateoΓCA (USA)Γ1991.

[RS93] E. Rose et A. Segev. TOOSQL - a temporal object-oriented query language. Dans *Proc. of the 12th International Conference on the Entity-Relationship Approach (ER)*ΓArlingtonΓTX (USA)Γ1993.

[RS97] H. Riedel et M.H. Scholl. A formalization of ODMG queries. Dans *Proc. of the 7th Int. Conference on Data Semantics (DS-7)*Γ LeysinΓ SwitzerlandΓ October 1997.

[SA85] R. T. Snodgrass et I. Ahn. A taxonomy of time in databases. Dans *Proc. of the ACM SIGMOD Int. Conference on Management of Data*ΓMay 1985.

[SBJS98] R.T. SnodgrassΓ M. BöhlenΓC. Jensen et A. Steiner. Transitioning temporal support in TSQL2 to SQL3. In Etzion et al. [EJS98].

[SC91] S. Su et H.-H. Chen. A temporal knowledge representation model OSAM/T and its query langage OQL/T. Dans *Proc. of the 17th Int. Conference on Very Large Databases (VLDB)*ΓBarcelonaΓSpainΓSeptember 1991.

[Sci91] E. Sciore. Multidimensional versioning for object-oriented databases. Dans *Proc. of the Int. Conference on Deductive and Object-Oriented Databases (DOOD)*Γ MunichΓGermanyΓDecember 1991. Springer-Verlag.

[Sci94] E. Sciore. Versioning and configuration management in an object-oriented data model. *The VLDB Journal*Γ3:77–106Γ1994.

[SDDM95] D. SchmidtΓA. Kotz DittrichΓW. Dreyer et R. Marti. Time-series: a neglected issue in temporal database research? In Clifford and Tuzhilin [CT95].

[Seg93] A. Segev. Join processing and optimization in temporal relational databases. In Tansel et al. [TCG+93].

[SFC98] P.-C. SchollΓM.-C. Fauvet et J.-F. Canavaggio. Un modèle d'historique pour un SGBD temporel. *TSI*Γ17(3)Γmars 1998.

[Sha99] D. Shasha. Time Series in Finance: the array database approach . http://cs.nyu.edu/cs/faculty/shasha/papers/jagtalk.htmlΓApril 1999.

[Shi81] D.W. Shipman. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*Γ6(1):140–173Γ1981.

[SJLL00] S. SaltenisΓC.S. JensenΓS. Leutenegger et M.A. López. Indexing the position of continuously moving objects. Dans *Proc. of the ACM SIGMOD Conference on Management of Data*ΓDallasΓTX (USA)ΓMay 2000.

[Skj97] B. Skjellaug. Temporal data : Time and object databases. Research Report 245Γ University of OsloΓDepartment of InformaticsΓApril 1997.

[SLR96] P. SeshadriΓM. Livny et R. Ramakrishnan. The design and implementation of a sequence database system. Dans *Proc. of the Int. Conference on Very Large Databases (VLDB)*ΓBombayΓIndiaΓ1996.

[Sno95a] R. T. Snodgrass. Temporal object-oriented databases: a critical comparison. Dans W. KimΓéditeurΓ*Modern database systems. The object model, interoperability and beyond*Γchapitre 19. Addison WesleyΓ1995.

[Sno95b] R. T. SnodgrassΓéditeur. *The TSQL2 temporal query language*. Kluwer Academic PublishersΓ1995.

[Sno99] R.T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan KaufmannΓJuly 1999.

[SS93] A. Segev et A. Shoshani. A temporal data model based on time sequences. In Tansel et al. [TCG+93].

[SSJ94] M.D. SooΓR.T. Snodgrass et C.S. Jensen. Efficient evaluation of the valid-time natural join. Dans *Proceedings of the 10th International Conference on Data Engineering (ICDE)*Γpages 282 – 292Γ1994.

[SSV98] A. SotiropoulouΓM. Souillard et C. Vassilakis. Temporal extension to ODMG. Dans *Proc. of the Workshop on Issues and Applications of Database Technology (IADT)*ΓBerlinΓGermanyΓJuly 1998.

[ST99] B. Salzberg et V.J. Tsotras. Comparison of access methods for time-evolving data. *ACM Computing Surveys*Γ31(2):158 – 221Γ1999.

[SVP+95] M. SchollΓA. VoisardΓJ.-P. PelouxΓL. Raynal et P. Rigaux. *SGBD Géographiques: Spécificités*. International Thompson PublishingΓ1995.

[TCG+93] A. U. TanselΓJ. CliffordΓS. GadiaΓS. JajodiaΓA. Segev et R. SnodggrassΓéditeurs. *Temporal Databases*. The Benjamin/Cummings Publishing CompanyΓ1993.

[Tom98] D. Toman. Point-based temporal extensions of SQL and their efficient implementation. In Etzion et al. [EJS98].

[TOO96a] TOOBIS ESPRIT Project. TODMΓ specification and design. Deliverable T31TR.1ΓMATRA CAP Systèmes – $0_2$ TechnologyΓDecember 1996.

[TOO96b] TOOBIS ESPRIT Project. TOQL specification. Deliverable T33TR.1ΓUniversity of AthensΓNational Technical University of AthensΓ01-PLIFORIKI S.A.Γ$0_2$ TechnologyΓDecember 1996.

[TOO97] TOOBIS ESPRIT Project. TOOBIS – temporal object-oriented databases in information systems. http://www.di.uoa.gr/toobisΓ1996 - 1997.

[TPP95] B. TheodoulidisΓP. Papapanagiotou et V. Pappas-Katsiafas. Interactive querying and visualisation in temporal databases. Dans *Workshop of the International Conference on Deductive and Object Oriented Databases (DOOD)*ΓSingaporeΓDecember 1995.

[TS97] B. Theodoulidis et M. Svinterikou. Temporal Unified Modeling Language (TUML). Rapport de recherche CH-97-11ΓCHOROCHRONOS TMR Research Network ProjectΓ1997.

[TT98] A.U. Tansel et E. Tin. Expressive power of temporal relational query languages and temporal completeness. In Etzion et al. [EJS98].

[Tuf84] E. Tufte. *The visual display of quantitative information*. Graphics PressΓ1984.

[Vod97] D. Vodislav. A visual programming model for user interface animation. Dans *Proc. Of The IEEE Int. Symposium On On Visual Languages (VL)*Γ1997.

[WD93] G.T.J. Wuu et U. Dayal. A uniform model for temporal and versioned object-oriented databases. In Tansel et al. [TCG+93].

[WDG95] R. WeissΓA. Duda et D.K. Gifford. Composition and search with a video algebra. *IEEE Multimedia*Γ2(1):12 – 25Γ1995.

[WJS95] X. S. Wang, S. Jajodia et V. S. Subrahmanian. Temporal modules : an approach toward federated temporal databases. *Information Systems*, 82(1–2):103 – 128, 1995.

[WJSW98] Y. Wu, S. Jajodia et X. Sean Wang. Temporal database bibliography update. In Etzion et al. [EJS98].

[WXCJ98] O. Wolfson, B. Xu, S. Chamberlain et L. Jiang. Moving objects databases: Issues and solutions. Dans *Proceedings of the 10th Int. Conference on Scientific and Statistical Database Management*, Capri, Italy, July 1998.

[ZCF+97] C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V. Subrahmanian et R. Zicari. Temporal databases. Dans *Advanced database systems*, chapitre 5 – 7. Morgan Kauffman, 1997.

[Zlo75] M. Zloof. Query by example. Dans *Proc. of the National Computer Conference*. American Federation for Information Processing Societies, 1975.

[Zur97] Thomas Zurek. *Optimisation of Partitioned Temporal Joins*. PhD thesis, University of Edinburgh, UK, November 1997.

# Annexe A

# Le type historique

Dans cette annexeΓnous spécifions le type historique et ses opérateurs. Certains de ces opérateurs étant d'ordre supérieur (au sens où ils prennent des fonctions en argument)Γleur spécification précise requiert un système de types qui traite les fonctions comme des citoyens à part entière. AussiΓnous décrivons dans un premier temps les signatures de ces opérateurs au moyen d'une notation inspirée des langages fonctionnels [CM95]Γavant de montrer comment ces spécifications peuvent être adaptées au système de types de l'ODMGΓau détriment du typage statique.

## A.1 Spécification fonctionnelle

### Notations

- T1 → T2 dénote le type des fonctions à domaine dans T1 et à valeur dans T2.
- {T} et [T] dénotent respectivement le type des ensembles et des listes dont les éléments sont de type T.
- ⟨T1, T2, ... Tn⟩ est le type des n-uplets dont le $i^{ème}$ composant est de type Ti ($1 \leq i \leq n$). Les attributs du type n-uplet peuvent être nommés au moyen de la notation ⟨L1: T1, L2: T2, ... Ln: Tn⟩.
- ⟨V1, V2, ... , Vn⟩ dénote une valeur de type n-uplet dont le $i^{ème}$ composant est Vi. Les composants peuvent être nommés avec la notation ⟨L1: V1, L2: V2, ... , Ln: Vn⟩.
- _ OP _ indique que l'opérateur binaire OP est utilisé avec une notation infixe.
- λx1Γx2Γ... Γxn • E dénote une valeur de type fonction : x1Γx2Γ... xn étant les paramètresΓet E l'expression fonctionnelle la décrivant.

### Sélecteurs

Type Historique<T> = Instant → T

Domaine: Historique<T> → EDI

Image: Historique<T> → { T }

Valeur: Historique<T>, Instant → T /* *Valeur(H, I)* = *valeur de l'historique H à l'instant I* */
/* *précondition: I* ∈ *Domaine(H)* */

Granularité: Historique<T> → Granularité /* *Granularité(H) = Granularité(Domaine(H))* */

Durée: Historique<T > → Durée /* *Durée(H) = Durée(Domaine(H))* */

Min: Historique<T : Numérique [1]> → T /* *Min(H) = Min(Image(H))* */

Max: Historique<T : Numérique> → T /* *Max(H) = Max(Image(H))* */

Sum: Historique<T : Numérique> → T

Avg: Historique<T : Numérique> → double

## Représentation par chroniques

IHistorique : [⟨moment : Instant, valeur : T⟩] → Historique<T>

/* *précondition: soit* $[IS_1, \ldots, IS_n]$ *le paramètre d'un appel à IHistorique :*
$\forall k \in [1..n-1]$ $(Granularité(IS_k.moment) = Granularité(IS_{k+1}.moment) \wedge IS_k.moment < IS_{k+1}.moment)$ */

XHistorique : [⟨moment : Interval, valeur : T⟩] → Historique<T>

/* *précondition: soit* $[XS_1, \ldots, XS_n]$ *le paramètre d'un appel à XHistorique:*
$\forall k \in [1..n\text{-}1]$ $(Granularité(XS_k.moment) = Granularité(XS_{k+1}.moment) \wedge XS_k.moment < XS_{k+1}.moment \wedge (XS_k.moment\ jouxte^{2}\ XS_{k+1}.moment \Rightarrow XS_k.valeur \neq XS_{k+1}.valeur))$ */

IChronique: Historique<T> → [⟨moment : Instant, valeur : T⟩]

/* $IHistorique(IChronique(H)) = H$ */

XChronique: Historique<T> → [⟨moment : Interval, valeur : T ⟩]

/* $XHistorique(XChronique(H)) = H$ */

## Opérateurs algébriques

*Restriction et transformation*

_ $\Sigma_{im}$ _: Historique<T>, (T → boolean) → Historique<T>

/* $H\ \Sigma_{im}\ P = \{\langle I,v\rangle \mid \langle I,v\rangle \in H \wedge P(v)\}$ */

_ $\Sigma_{dom}$ _: Historique<T>, TSequence → Historique<T>

/* $H\ \Sigma_{dom}\ S = \{\langle I,v\rangle \mid \langle I,v\rangle \in H \wedge I \in S\}$ */

Map: Historique<T>, (T → T') → Historique<T'>

/* $Map(H, f) = \{\langle I, f(v)\rangle \mid \langle I,v\rangle \in H\}$ */

1. La notation T : Numérique est utilisée pour exprimer que T est l'un des types suivants : short, long, unsigned short, unsigned long, float ou double.
2. Deux intervalles se jouxtent s'ils ne s'intersectent pas et si leur union est aussi un intervalle.

*Produit interne et externe*

_ $*_{\cap}$ _: Historique<T1>, Historique<T2> $\rightarrow$ Historique<$\langle$T1,T2$\rangle$>
/* $H1 *_{\cap} H2 = \{\langle I, \langle v1, v2\rangle\rangle \mid \langle I, v1\rangle \in H1 \wedge \langle I, v2\rangle \in H2\}$ */
/* *précondition: Granularité(H1) = Granularité(H2)* */
_ $*_{\cup}$ _: Historique<T1>, Historique<T2> $\rightarrow$ Historique<$\langle$T1,T2$\rangle$>
/* $H1 *_{\cup} H2 = H1 *_{\cap} H2 \cup \{\langle I,\langle v1, Nil^3\rangle\rangle \mid \langle I,v1\rangle \in H1 \wedge I \notin Domaine(H2)\}$
$\cup \{\langle I,\langle Nil, v2\rangle\rangle \mid \langle I,v2\rangle \in H2 \wedge I \notin Domaine(H1)\}$ */
/* *précondition: Granularité(H1) = Granularité(H2)* */

*Opérateurs ensemblistes*

_ $\cap$ _: Historique<T>, Historique<T> $\rightarrow$ Historique<T>
/* $H1 \cap H2 = \{ \langle I, v\rangle \mid \langle I, v\rangle \in H1 \wedge \langle I, v\rangle \in H2 \}$ */
_ $\setminus$ _: Historique<T>, Historique<T> $\rightarrow$ Historique<T>
/* $H1 \setminus H2 = \{ \langle I, v\rangle \mid \langle I, v\rangle \in H1 \wedge \langle I, v\rangle \notin H2 \}$ */
/* *précondition: Granularité(H1) = Granularité(H2)* */
_ $\cup_{+}$ _: Historique<T>, Historique<T> $\rightarrow$ Historique<T>
/* $H1 \cup_{+} H2 = \{ \langle I, v\rangle \mid \langle I, v\rangle \in H2 \vee (\langle I, v\rangle \in H1 \wedge I \notin Domaine(H2)) \}$ */
/* *précondition: Granularité(H1) = Granularité(H2)* */

*Agrégation cumulative*

CMin: Historique<T : Numérique> $\rightarrow$ Historique<T>
/* $CMin(H) = \{ \langle i, m\rangle \mid i \in Domaine(H) \wedge$
$m = Min(Image(H \Sigma_{dom} [Min(Domaine(H))..i]))\}$ */
CMax: Historique<T : Numérique> $\rightarrow$ Historique<T>
/* $CMax(H) = \{ \langle i, m\rangle \mid i \in Domaine(H) \wedge$
$m = Max(Image(H \Sigma_{dom} [Min(Domaine(H))..i]))\}$ */
CSum: Historique<T : Numérique> $\rightarrow$ Historique<T>
/* $CSum(H) = \{ \langle i, s\rangle \mid i \in Domaine(H) \wedge$
$s = Sum(\{ v \mid \langle I, v\rangle \in (H \Sigma_{dom} [Min(Domaine(H))..i])\})\}$ */
CAvg: Historique<T : Numérique> $\rightarrow$ Historique<double>
/* $CAvg(H) = \{ \langle i, s\rangle \mid i \in Domaine(H) \wedge$
$s = Avg(\{ v \mid \langle I, v\rangle \in (H \Sigma_{dom} [Min(Domaine(H))..i])\})\}$ */

*Regroupement*

_ $\Upsilon$ _: Historique<T>, Granularité $\rightarrow$ Historique<Historique<T>>
/* $H \Upsilon G =$
$\{ \langle I, H'\rangle \mid \exists I' \in Domaine(H), approx(I', G) = I \wedge H' = H \Sigma_{dom} expand(I, Granularité(H))\}$
*précondition:* $Granularité(H) \prec G$ */

3. Nil dénote la valeur neutre du type en question.

_ Θ _: Historique<T>, ⟨ Durée, Durée ⟩ → Historique<Historique<T>>
/* $H \Theta \langle D1, D2 \rangle =$
$\{ \langle I, H' \rangle \mid [I - D1..I + d] \subseteq Domaine(H) \wedge H' = H \Sigma_{dom} [I - D1..I + D2] \}$
*préconditions:*

- *Mesure(D1) > 0 et Mesure(D2) > 0.*
- *Granularité(D1) = Granularité(H)*
  *ou Granularité(D1) régulière par rapport à Granularité(H) (Cf. § 3.1.1) et idem pour D2.*

*/

*Découpage*

AprèsPremier: Historique<T>, (T → boolean) → Historique<T>
/* *AprèsPremier(H, P)* $= \{ \langle I, v \rangle \mid \langle I, v \rangle \in H \wedge \exists \langle I', v' \rangle \in H\ (P(v') \wedge I \geq I') \}$ */
AvantPremier: Historique<T>, (T → boolean) → Historique<T>
/* *AvantPremier(H, P)* $= \{ \langle I, v \rangle \mid \langle I, v \rangle \in H \wedge \neg\exists \langle I', v' \rangle \in H\ (P(v') \wedge I \geq I') \}$ */
AprèsDernier: Historique<T>, (T → boolean) → Historique<T>
/* *AprèsDernier(H, P)* $= \{ \langle I, v \rangle \mid \langle I, v \rangle \in H \wedge \neg\ \exists \langle I', v' \rangle \in H\ (P(v') \wedge I \leq I') \}$ */
AvantDernier: Historique<T>, (T → boolean) → Historique<T>
/* *AvantDernier(H, P)* $= \{ \langle I, v \rangle \mid \langle I, v \rangle \in H \wedge \exists \langle I', v' \rangle \in H\ (P(v') \wedge I \leq I') \}$ */

## A.2 Spécification en ODMG

Ci-après nous traduisons les spécifications fonctionnelles de la section précédente en une interface ODMG. Le système de types de l'ODMG n'étant pas d'ordre supérieur le problème de la traduction des opérateurs comportant des paramètres de type fonctionnel se pose. Nous avons choisi de coder ces paramètres sous forme d'une chaîne de caractères et d'une liste d'objets. La chaîne de caractères décrit une requête OQL où peuvent figurer les symboles $$ $1 $2 etc. Le symbole $$ joue le rôle du paramètre de la fonction codée par la requête[4]. La liste d'objets quant à elle permet d'associer une valeur à chacun des symboles $1 $2 etc. figurant dans la requête. Elle définit ainsi le contexte d'évaluation de la fonction codée.

```
interface Historique<T> {
  EDI Domaine();
  set<T> Image();
  T Valeur(Instant);
  Granularité Granularité();
  Durée Durée();
```

4. On remarque que toutes les fonctions apparaissant en paramètre d'un opérateur sur les historiques sont unaires.

```
  set<moment: Instant, valeur: T> IChronique();
  set<moment: Intervalle, valeur: T> XHistorique();
  Historique<T> Rsi(string, list(Object));
  Historique<T> Ren(EDI);
  Historique<Object> Map(string, list(Object));
  Historique<struct<gauche: T, droite: Object>> ProdI(Historique<Object>);
  Historique<struct<gauche: T, droite: Object>> ProdE(Historique<Object>);
  Historique<T> Intersection(Historique<T>);
  Historique<T> Différence(Historique<T>);
  Historique<T> UnionPlus(Historique<T>);
  Historique<Historique<T>> GroupementGranularité(Granularité);
  Historique<Historique<T>> GroupementDurée(Granularité, Durée, Durée);
  Historique<T> ApresPremier(string, list(Object));
  Historique<T> AvantPremier(string, list(Object));
  Historique<T> ApresDernier(string, list(Object));
  Historique<T> AvantDernier(string, list(Object));
}
interface Numérique[5] {
  boolean Inferieur(Numérique);
  Numérique Plus(Numérique);
  double Division(Numérique);
}
interface HistoriqueNumérique<T : Numérique> : Historique {
  /* cette interface spécialise l'interface Historique */
  T Min();
  T Max();
  T Sum();
  double Avg();
  Historique<T> CMax();
  Historique<T> CMin();
  Historique<T> CSum();
  Historique<double> CAvg();
}
```

Dans la définition de cette interface, nous faisons l'hypothèse que l'ODMG supporte la définition de classes paramétrées contraintes (Cf. § 2.1.2). Or, même si des interfaces

5. On suppose que les classes short, long, unsigned short, unsigned long, float et double prédéfinies en ODMG implantent cette interface.

paramétrées apparaissent dans la passerelle pour C++Γelles sont absentes des passerelles pour Java et SmallTalkΓet elles ne figurent pas dans la syntaxe d'ODL[6]

Pour effectuer le passage de l'interface paramétrée ci-dessus à une interface non-paramétréeΓ on peut remplacer toutes les occurrences du paramètre TΓpar le type Object. Pour les besoins d'une application particulièreΓil serait éventuellement possible par la suite de spécialiser cette interface "générique"Γpour manipuler des historiques d'un type spécifique (p. ex. des historiques d'employésΓde lieuxΓetc.). Cette approche revient en fait à coder le mécanisme des classes paramétrées au moyen de l'héritage.

6. Dans la syntaxe d'ODL, figurent un certain nombre de générateurs de types de littéraux (p. ex. list, bag et struct). Toutefois, le standard n'envisage pas la possibilité de pouvoir rajouter des nouveaux générateurs de types en plus de ceux qu'il fournit.

# Annexe B

# TempOQL

## B.1 Syntaxe et sémantique

Dans les paragraphes suivants, nous étendons la formalisation d'OQL proposée dans [RS97] pour y inclure les opérateurs propres à TempOQL. La formalisation de chaque opérateur est composée de trois parties :

- **Syntaxe.** Dans une notation de style BNF où les terminaux sont typographiés dans une police sans empattements.
- **Typage.** Dans la notation $\frac{\text{prémisse}}{\text{implication}}$. L'expression q::t établit que la requête q est de type t, alors que q[x::t']::t établit que q est de type t modulo l'hypothèse que x est de type t'.
- **Sémantique.** En termes d'équations récursives faisant intervenir des opérateurs du modèle d'historiques. La sémantique de chaque requête est paramétrée par une fonction de valuation qui fixe la valeur des symboles "libres" dans la requête. L'expression $\nu[\mathsf{x} \leftarrow \mathsf{v}]$ dénote la valuation identique à $\nu$ modulo le fait qu'elle associe la valeur v au symbole x. Sauf indication contraire, les préconditions définies au niveau des opérateurs du modèle d'historiques (en particulier les contraintes liant les granularités des arguments) sont héritées par les opérateurs TempOQL qu'ils servent à définir.

Les opérateurs sur les valeurs temporelles étant décrites dans [CD97], nous nous limitons par la suite aux opérateurs sur les historiques.

### Valeur d'un historique à un instant

Syntaxe : <query> ::= <query> '[' <query> ']'

Typage : $$\frac{q_1 :: \text{Historique}<\tau>,\ q_2 :: \text{Instant}}{q_1[q_2] :: \tau}$$

Sémantique : $[\![ q_1[q_2] ]\!]_\nu = \text{Valeur}\ ([\![ q_1 ]\!]_\nu, [\![ q_2 ]\!]_\nu)$

**Remarque.** Le symbole 'query' est l'un des non-terminaux de la grammaire d'OQL telle que spécifiée dans la version 2.0 du standard ODMG [CB97]. Dans la version 3.0Γla grammaire d'OQL est spécifiée en utilisant d'autres non-terminauxΓtout en restant équivalente à celle de la version 2.0. Modulo sa réformulationΓla grammaire de TEMPOQL que nous spécifions ici peut donc être intégrée dans la nouvelle version du standard.

### Restriction suivant le domaine

Syntaxe : <query> ::= <query> during <query>

Typage : $$\frac{q_1 :: \text{Historique}<\tau_1>,\ q_2 :: \text{Instant} \vee q_2 :: \text{EDI}}{q_1 \text{ during } q_2 :: \text{Historique}<\tau_1>}$$

Sémantique: $[\![ q_1 \text{ during } q_2 ]\!]_\nu =$

$$\begin{cases} [\![ q_1 ]\!]_\nu \ \Sigma_{\text{dom}} \ [\![ q_2 ]\!]_\nu & \text{si } \tau_2 \text{ sous-type de EDI} \\ [\![ q_1 ]\!]_\nu \ \Sigma_{\text{dom}} & \text{si } \tau_2 \text{ sous-type de Instant et} \\ \text{expand } ([\![ q_2 ]\!]_\nu \Gamma \text{Granularité } ([\![ q_1 ]\!]_\nu)) & \text{Granularité } ([\![ q_1 ]\!]_\nu) \prec \text{Granularité } ([\![ q_2 ]\!]_\nu) \end{cases}$$

Dans cette règleΓla précondition suivante s'impose :

$\tau_2$ sous-type de EDI $\vee$ ($\tau_2$ sous-type de Instant $\wedge$ Granularité $([\![ q_1 ]\!]_\nu) \prec$ Granularité $([\![ q_2 ]\!]_\nu)$).

### Restriction suivant l'image

Syntaxe : <query>::= <query> as <identifier> when <query>

Typage : $$\frac{q_1 :: \text{Historique}<\tau>,\ q_2[x :: \tau] :: \text{boolean}}{q_1 \text{ as x when } q_2 :: \text{Historique}<\tau>}$$

Sémantique : $$\begin{aligned} [\![ q_1 \text{ as x when } q_2 ]\!]_\nu &= [\![ q_1 ]\!]_\nu \ \Sigma_{\text{im}} \ \lambda v \bullet [\![ q_2 ]\!]_{\nu[x \leftarrow v]} \\ &= \{ \langle I \Gamma v \rangle \mid \langle I \mathbb{V} \rangle \in [\![ q_1 ]\!]_\nu \wedge [\![ q_2 ]\!]_{\nu[x \leftarrow v]} \} \end{aligned}$$

### Transformation d'un historique

Syntaxe : <query> ::= map <query> on <query> as <identifier> when <query>

Typage : $$\frac{q_2 :: \text{Historique}<\tau_2>,\ q_1[x :: \tau_2] :: \tau_1,\ q_3[x :: \tau_2] :: \text{boolean}}{\text{map } q_1 \text{ on } q_2 \text{ as x when } q_3 :: \text{Historique}<\tau_1>}$$

Sémantique :

$$\begin{aligned} [\![ \text{map } q_1 \text{ on } q_2 \text{ as x when } q_3 ]\!]_\nu &= \text{Map } ([\![ q_2 ]\!]_\nu \ \Sigma_{\text{im}} \ \lambda v \bullet [\![ q_3 ]\!]_{\nu[x \leftarrow v]} \Gamma \lambda w \bullet [\![ q_1 ]\!]_{\nu[x \leftarrow w]}) \\ &= \{ \langle I \Gamma [\![ q_1 ]\!]_{\nu[x \leftarrow v]} \rangle \mid \langle I \mathbb{V} \rangle \in [\![ q_2 ]\!]_\nu \wedge [\![ q_3 ]\!]_{\nu[x \leftarrow v]} \} \end{aligned}$$

### Produit d'historiques

Syntaxe : <query>::= join (<identifier> : <query> {Γ<identifier> : <query> } >)

Typage : $$\frac{q_1 :: \text{Historique}<\tau_1>,\ q_2 :: \text{Historique}<\tau_2>,\ \ldots,\ q_n :: \text{Historique}<\tau_n>}{\text{join } (l_1 : q_1, l_2 : q_2, \ldots, l_n : q_n) :: \text{Historique}<\text{struct}<l_1 : \tau_1, l_2 : \tau_2, \ldots l_n : \tau_n>>}$$

Sémantique : $[\![ \text{join } (l_1 : q_1, l_2 : q_2, \ldots\ l_n : q_n) ]\!]_\nu = [\![ q_1 ]\!]_\nu *_\cap [\![ q_2 ]\!]_\nu *_\cap \ldots *_\cap [\![ q_n ]\!]_\nu$

Le produit externe (ojoin) est défini de façon analogue.

### Généralisation

*Généralisation des opérateurs arithmétiques à deux historiques d'entiers*

Soit $\theta \in \{ +, -, *, \text{div}, \text{mod} \}$; $\tau \in \{\text{long}, \text{short}, \text{unsigned long}, \text{unsigned short}\}$

Syntaxe : <query> ::= <query> $\theta$ <query>

Typage : $\dfrac{q_1 :: \text{Historique}<\tau>,\ q_2 :: \text{Historique}<\tau>}{q_1\ \theta\ q_2 :: \text{Historique}<\tau>}$

Sémantique : $[\![ q_1\ \theta\ q_2 ]\!]_\nu = \text{Map}\ ([\![ q_1 ]\!]_\nu *_\cap [\![ q_2 ]\!]_\nu, \lambda\langle v1, v2 \rangle \bullet v1\ \theta\ v2)$

*Généralisation des opérateurs arithmétiques à un historique d'entiers et un entier*

Soit $\theta \in \{ +, -, *, \text{div}, \text{mod} \}$; $\tau \in \{\text{long}, \text{short}, \text{unsigned long}, \text{unsigned short}\}$

Syntaxe: <query> ::= <query> $\theta$ <query>

Typage : $\dfrac{q_1 :: \text{Historique}<\tau>,\ q_2 :: \tau}{q_1\ \theta\ q_2 :: \text{Historique}<\tau>}$

Sémantique : $[\![ q_1\ \theta\ q_2 ]\!]_\nu = \text{Map}\ ([\![ q_1 ]\!]_\nu, \lambda v \bullet v\ \theta\ [\![ q_2 ]\!]_\nu)$

La généralisation des opérateurs arithmétiques à un entier et un historique d'entiers s'obtient par symétrie. De même, d'autres spécifications analogues à ces deux dernières définissent la généralisation aux historiques de tous les autres opérateurs d'OQL, à l'exception de l'opérateur d'application d'une propriété temporelle à un objet (Cf. § 4.1.2).

*Généralisation de l'application d'une propriété de type historique*

Dans le méta-modèle de l'ODMG [CB00], les propriétés sont des instances de l'interface Property. Cette méta-classe possède en particulier une propriété nommée type qui retrouve le type de l'image d'une propriété. Les types sont eux-mêmes modélisés comme des instances d'une méta-interface Type, sur laquelle est définie en particulier une propriété nommée properties de type set<Property>.

Syntaxe[1] : <query> ::= <query> <dot> <identifier>

<dot> ::= . | ->

Typage : $\dfrac{q :: \text{Historique}<\tau_1>,\ P \in \tau_1.\text{properties},\ P.\text{type} = \text{Historique}<\tau_2>}{q.P :: \text{Historique}<\tau_2>}$

Sémantique : $[\![ q.P ]\!]_\nu = \{ \langle i, o' \rangle \mid \langle i, o \rangle \in [\![ q ]\!]_\nu \wedge \langle i, o' \rangle \in o.P \}$

La généralisation de l'appel de méthodes se généralise de façon analogue.

1. Ces deux règles font partie de la grammaire d'OQL v2.0 [CB97].

### Découpage

Syntaxe: <query>::= <query> as <identifier> afterfirst <query>

Typage : $\dfrac{q_1 :: \text{Historique}<\tau>,\ q_2[x :: \tau] :: \text{boolean}}{q_1 \text{ as x afterfirst } q_2 :: \text{Historique}<\tau>}$

Sémantique :

$$[\![ q_1 \text{ as x afterfirst } q_2 ]\!]_\nu = \text{AprèsPremier}([\![ q_1 ]\!]_\nu \Gamma \lambda v \bullet [\![ q_2 ]\!]_{\nu[x \leftarrow v]})$$
$$= \{\langle I \Gamma v \rangle \mid \langle I \Gamma v \rangle \in [\![ q_1 ]\!]_\nu \wedge \exists \langle I' \Gamma v' \rangle \in [\![ q_1 ]\!]_\nu \ ([\![ q_2 ]\!]_{\nu[x \leftarrow v']} \wedge I \geq I')\}$$

Les constructions beforefirstΓafterlast et beforelast sont définies de façon similaire.

### Regroupement

*Avec un seul paramètre de regroupement*

Syntaxe: <query> ::= map <query> on <query> as <identifier> when <query>
group by <query> having<query>

$$q_2 :: \text{Historique}<\tau_2>,\ \ q_1[\text{partition} :: \text{Historique}<\tau_2>] :: \tau_1,\ q_3[x :: \tau_2] :: \text{boolean}$$
$$q_4 :: \text{Granularité} \vee q_4 :: \text{Durée},\ q_5[\text{partition} :: \text{Historique}<\tau_2>] :: \text{boolean}$$

Typage : $\dfrac{}{\text{map } q_1 \text{ on } q_2 \text{ as x when } q_3 \text{ group by } q_4 \text{ having } q_5 :: \mathit{Historique}<\tau_1>}$

Sémantique : $[\![ \text{map } q_1 \text{ on } q_2 \text{ as x when } q_3 \text{ group by } q_4 \text{ having } q_5 ]\!]_\nu =$

$$\begin{cases} \text{Map}(\ (([\![ q_2 ]\!]_\nu\ \Sigma_{\text{im}}\ \lambda v \bullet [\![ q_3 ]\!]_{\nu[x \leftarrow v]})\ \Gamma\ [\![ q_4 ]\!]_\nu) \\ \qquad \Sigma_{\text{im}}\ \lambda w \bullet [\![ q_5 ]\!]_{\nu[\text{partition} \leftarrow w]} \Gamma \lambda y \bullet [\![ q_1 ]\!]_{\nu[\text{partition} \leftarrow y]}) & \text{si } q_4 \text{ de type Granularité} \\ \text{Map}(\ (([\![ q_2 ]\!]_\nu\ \Sigma_{\text{im}}\ \lambda v \bullet [\![ q_3 ]\!]_{\nu[x \leftarrow v]})\ \Delta\ \langle [\![ q_4 ]\!]_\nu \Gamma 0 \rangle) \\ \qquad \Sigma_{\text{im}}\ \lambda w \bullet [\![ q_5 ]\!]_{\nu[\text{partition} \leftarrow w]} \Gamma \lambda y \bullet [\![ q_1 ]\!]_{\nu[\text{partition} \leftarrow y]}) & \text{si } q_4 \text{ de type Durée} \end{cases}$$

*Avec deux paramètres de regroupement*

Syntaxe: <query> ::= map <query> on <query> as <identifier> when <query>
group by <query> , <query> having<query>

$$q_2 :: \text{Historique}<\tau_2>,\ \ q_1[\text{partition} :: \text{Historique}<\tau_2>] :: \tau_1,\ q_3[x :: \tau_2] :: \text{boolean}$$
$$q_4 :: \text{Durée}, q_5 :: \text{Durée}\ q_6[\text{partition} :: \text{Historique}<\tau_2>] :: \text{boolean}$$

Typage : $\dfrac{}{\text{map } q_1 \text{ on } q_2 \text{ as x when } q_3 \text{ group by } q_4, q_5 \text{ having } q_6 :: \mathit{Historique}<\tau_1>}$

Sémantique : $[\![ \text{map } q_1 \text{ on } q_2 \text{ as x when } q_3 \text{ group by } q_4 \text{ having } q_5 ]\!]_\nu =$

$$\text{Map}(\ (([\![ q_2 ]\!]_\nu\ \Sigma_{\text{im}}\ \lambda v \bullet [\![ q_3 ]\!]_{\nu[x \leftarrow v]})\ \Delta\ \langle [\![ q_4 ]\!]_\nu \Gamma [\![ q_5 ]\!]_\nu \rangle)\ \Sigma_{\text{im}}\ \lambda w \bullet [\![ q_6 ]\!]_{\nu[\text{partition} \leftarrow w]} \Gamma$$
$$\lambda y \bullet [\![ q_1 ]\!]_{\nu[\text{partition} \leftarrow y]})$$

## B.2 Langage de description de motifs

### Préliminaires

La sémantique du langage de description de motifs est définie de façon similaire à celle de la logique temporelle étendue (ETL) [Eme90]. Plus précisément, une relation dite de *satisfaction* lie les expressions du langage (vues comme des formules logiques) à des structures abstraites modélisant des instances de bases de données temporelles. Les structures considérées sont de la forme $< \mathcal{T}, \mathcal{S} >$ où $\mathcal{T}$ est un ensemble d'instants de même granularité, et $\mathcal{S}$ est une suite $\mathcal{T}$-indexée d'*interprétations* des symboles de constante, de fonction et de relation apparaissant dans une expression du langage.

Vis-à-vis d'une base de données temporelles, les symboles de fonction modélisent des propriétés du schéma. Dans le cas d'un symbole de fonction modélisant une propriété fugitive, son interprétation est la même quel que soit l'instant considéré. Dans le cas d'un symbole de fonction P modélisant une propriété temporelle à domaine dans une classe C, son interprétation à un instant i (notée $\mathcal{S}_i(P)$) est définie comme suit :

$$\forall o \in extent(C)\ \mathcal{S}_i(P(o)) = Valeur(o.P, i)$$

Où extent(C) est l'extension de la classe C [2].

Les symboles de relation de leur côté, sont ceux du langage de requêtes OQL : égalité, relations d'ordre, prédicat like sur les chaînes de caractères. Leur interprétation n'est dépendante ni de l'instant considéré, ni même de la base de données.

Enfin, les symboles de constante modélisent des noms persistants (c.à.d. des extensions de classes) et des constantes des types littéraux (entiers, réels, etc.).

### Sémantique de la relation de satisfaction

Étant donné un modèle $\mathcal{H} =< \mathcal{T}, \mathcal{S} >$, la satisfaction d'une formule bien formée f par rapport à deux instants i et j (j > i) dans $\mathcal{T}$ est notée $\mathcal{H}, i, j \models$ f. Intuitivement, $\mathcal{H}, i, j \models$ f signifie qu'il existe une occurrence du motif f entre les instants i et j.

La relation de satisfaction $\models$ est décrite par les équations récursives ci-après (f, f1 and f2 dénotent des formules bien formées).

1. $\mathcal{H}, i, i \models$ f (pour une formule atomique $f$) ssi $\mathcal{S}_i \models f$ selon la définition de $\models$ dans la logique des prédicats du premier ordre.

2. L'extension complète s'il s'agit d'une classe temporelle (Cf. 3.2.2).

2. $\mathcal{H}, i, j \models f1 \mid\mid f2$ ssi $\mathcal{H}, i, j \models f1$ ou $\mathcal{H}, i, j \models f2$.

3. $\mathcal{H}, i, j \models f1$ followed by $f2$ ssi il existe $k \in \mathcal{T}\Gamma k \geq i\Gamma$tel que $\mathcal{H}, i, k \models f1$ et $\mathcal{H}, k+1, j \models f2$.

4. $\mathcal{H}, i, j \models (f)^1$ ssi $\mathcal{H}, i, j \models f$.

5. $\mathcal{H}, i, j \models (f)^n$ $(n > 1)$ ssi il existe $k \geq i, k \in \mathcal{T}$ tel que $\mathcal{H}, i, k \models f$ et $\mathcal{H}, k+1, j \models (f)^{n-1}$.

6. $\mathcal{H}, i, j \models$ several $f$ ssi il existe $n \geq 1$ tel que $\mathcal{H}, i, j \models (f)^n$.

7. Soit $\theta \in \{<, \leq, >, \geq, =\}$. $\mathcal{H}, i, j \models f$ during $\theta$ $d$ ssi il existe $n \geq 1$ tel que $\mathcal{H}, i, j \models (f)^n$ et $(i - j)$ $\theta$ $d$.

On remarque que les règles 4 et 5 ne définissent pas des expressions valides du langage (comme c'est le cas de toutes les autres règles)Γmais servent uniquement comme intermédiaires à la spécification des opérateurs several et during.

En plus des opérateurs spécifiés par les règles ci-dessusΓle langage de description de motifs comporte les "racourcis" suivants (f1 et f2 étant des formules bien formées):

- * ≡ several true
- f1 during d ≡ f1 during = d

### Intégration du langage de description de motifs dans le modèle Tempos

Afin d'intégrer le langage de description de motifs au modèle d'historiques de TEMPOSΓ nous introduisons un opérateur à valeur booléenne dénommé RechercheMotif. IntuitivementΓ cet opérateur détermine si un historique comporte au moins une occurrence d'un motif donné.

RechercheMotif: History<T>, string → boolean

*/* RechercheMotif (H, P) – P dénote une expression du langage de description de motifs et il existe i, j ∈ Domaine (H), <<Domaine (H), $\mathcal{S}$>, i, j> ⊨ P. */*

Où $\mathcal{S}$ dénote la structure obtenue à partir de la base de données considéréeΓen suivant le procédé décrit ci-haut. Le motif P peut comporter une constante notée $$ dont l'interprétation à un instant est donnée par la valeur de l'historique H à cet instant.

L'intégration de l'opérateur ci-dessus dans l'interface ODMG décrite dans la section A.2Γ donne lieu à une méthode dont la signature est :

```
interface Historique<T> {
  ...
  boolean RechercheMotif(string, list(Object));
}
```

# Annexe C

# Visualisation point-par-point

Dans cette annexe, nous développons une modélisation conceptuelle en UML de la structure et le comportement des objets mis en jeu par la technique de navigation point-par-point (Cf. § 4.2.3). Cette modélisation a été exploitée lors de l'implantation de la technique de visualisation décrite dans la section 6.2.3. Elle a pour but de fournir une documentation en vue d'implantations futures.

## C.1 Modélisation de la fenêtre "droite temporelle"

Le diagramme de la figure C.1 décrit la structure de la fenêtre "droite-temporelle".

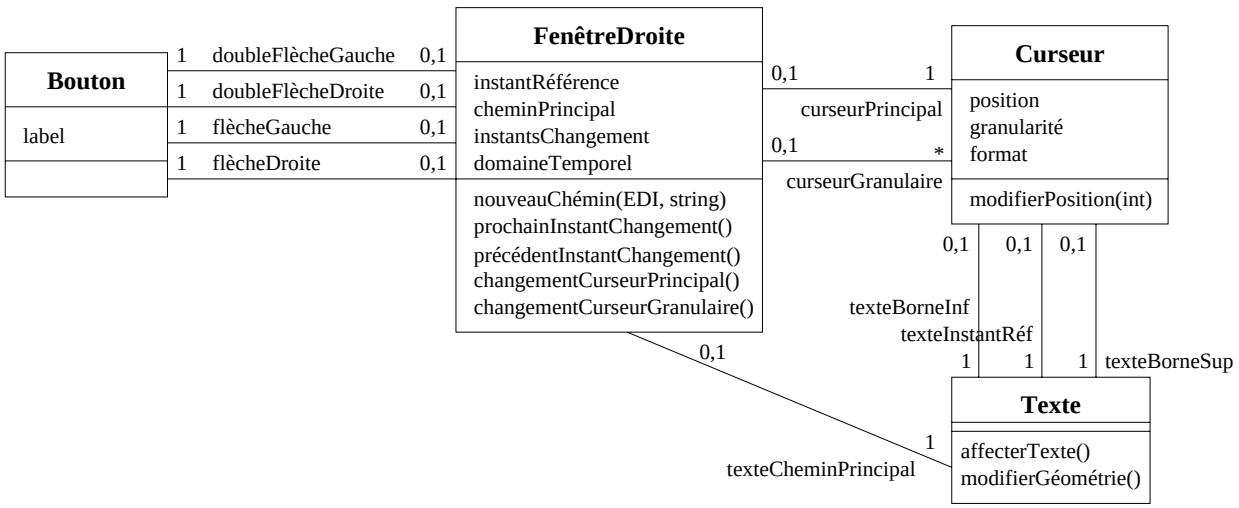



FIG. C.1 – *Diagramme de classes de la fenêtre "droite temporelle".*

En l'absence de curseurs granulaires, un objet de la classe FenêtreDroite est composé d'un curseur dit principal, de quatre boutons étiquetés par des flèches, et de quatre objets "textes" censés gérer l'affichage de l'instant de référence, des deux bornes du domaine temporel et du

chemin de navigation principal. Lorsque la fenêtre comporte des curseurs granulaires, chacun d'entre eux est associé à un objet de type "texte" servant à refléter sa position. Dans tous les cas, le contenu d'un objet texte attaché à un curseur est déterminé par un objet de type "format" attaché au curseur concerné.

La méthode prochainInstantChangement (resp. précédentInstantChangement) de la classe FenêtreDroite a pour rôle de déplacer le curseur jusqu'au plus petit (resp. plus grand) instant de changement supérieur ou égal (resp. inférieur ou égal) à l'instant de référence courant.

La méthode nouveauChemin, qui prend en arguments un ensemble d'instants et une chaîne de caractères, modélise le changement de chemin de principal. À cet effet il modifie les attributs cheminPrincipal et instantsChangement et propage ces changements vers l'objet de type Texte responsable de l'affichage du chemin principal.

Lorsqu'une fenêtre "droite-temporelle" comporte des curseurs granulaires, le comportement de ceux-ci est lié à celui du curseur principal. Plus précisément, lorsque l'utilisateur interagit avec l'un des curseurs granulaires, cet événement est notifié à l'objet de la classe FenêtreDroite correspondant. Cet objet calcule alors le nouvel instant de référence, et met à jour le curseur principal et les curseurs granulaires. De même, toute interaction sur le curseur principal est notifiée à l'objet de la classe FenêtreDroite correspondant, qui propage cette modification vers chacun des curseurs granulaires. La méthode changementCurseurPrincipal (resp. changementCurseurGranulaire) est utilisée pour notifier à la fenêtre-droite un changement survenu sur le curseur principal (resp. sur l'un des curseurs granulaires). Les diagrammes de collaboration apparaissant dans les figures C.2 et C.3 résument ces considérations.

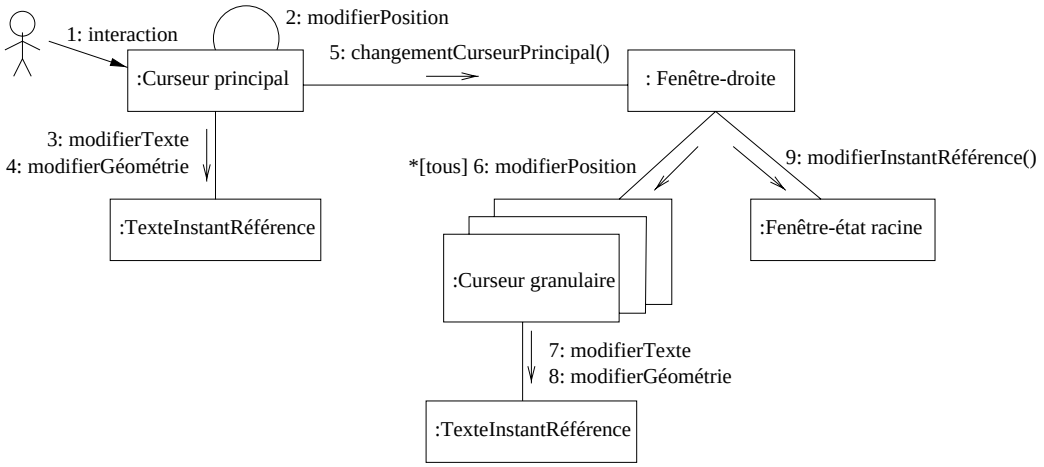



FIG. C.2 – *Diagramme de collaboration décrivant les actions consécutives à un déplacement du curseur principal.*

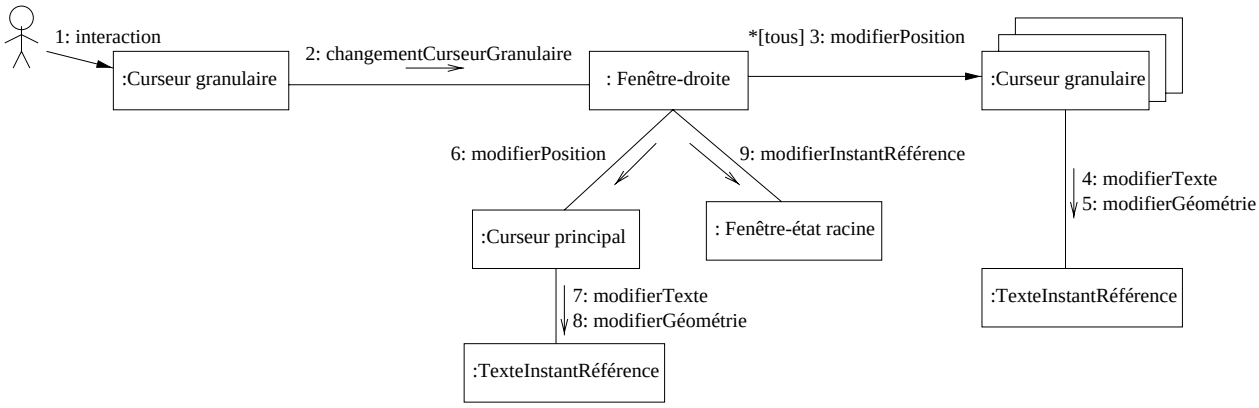



FIG. C.3 – *Diagramme de collaboration décrivant les actions consécutives à un déplacement d'un des curseurs granulaires.*

## C.2 Modélisation des fenêtres-états

Le diagramme de classes de la figure C.4 décrit la structure des fenêtres-états.

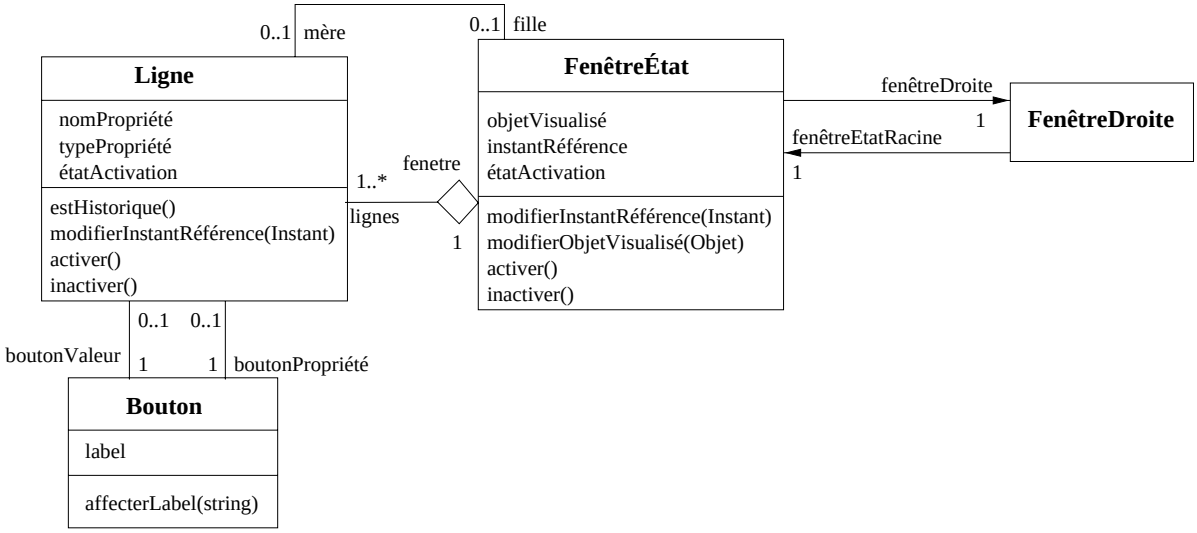



FIG. C.4 – *Diagramme de classes des fenêtres-états.*

Une fenêtre-état est composée d'une liste de "lignes"Γchacune dénotant une propriété de l'objet visualisé. Chaque ligne est elle-même composée de deux boutonsΓl'un contenant le nom de la propriété et l'autre sa valeur. Lorsque la valeur d'une propriété est un objet complexeΓl'utilisateur peut interagir avec le bouton affichant cette valeurΓce qui a pour effet de créer une nouvelle fenêtre-état. Une relation hiérarchique de type mère/fille se forme ainsi entre les fenêtres-états. L'utilisateur peut aussi interagir avec les boutons affichant des noms de propriété dans le but de modifier le chemin principal visualisé. Lorsqu'une telle interaction a lieuΓla ligne à laquelle appartient le bouton concernéΓcalcule les instants de changement

de sa valeur et transmet le résultat à la fenêtre “droite temporelle”.

Lorsqu'un changement survient dans la fenêtre “droite temporelle”Γle nouvel instant de référence est transmis à la fenêtre-état racineΓqui le transmet à son tour à chacune de ses lignes. Lorsqu'une ligne recevant cette notification est historique (c.à.d. que la propriété qu'elle dénote est temporelle)Γla valeur de cet attribut est recalculée. Par ailleursΓsi la ligne en question possède un filsΓla notification du changement d'instant de référence est propagé vers celui-ci. S'il arrive que suite à un changement de l'instant de référenceΓl'objet visualisé par une fenêtre-état devient “nul” (c.à.d. que le chemin qu'il dénote est indéfini à l'instant de référence)Γalors cette fenêtre-étatΓainsi que toutes les fenêtres-états qui dépendent transitivement d'elleΓdeviennent inactives. L'inactivation d'une fenêtre entraîne l'inactivation de chacune de ses lignes. Les diagrammes de collaboration de la figure C.5 résument ces considérations.

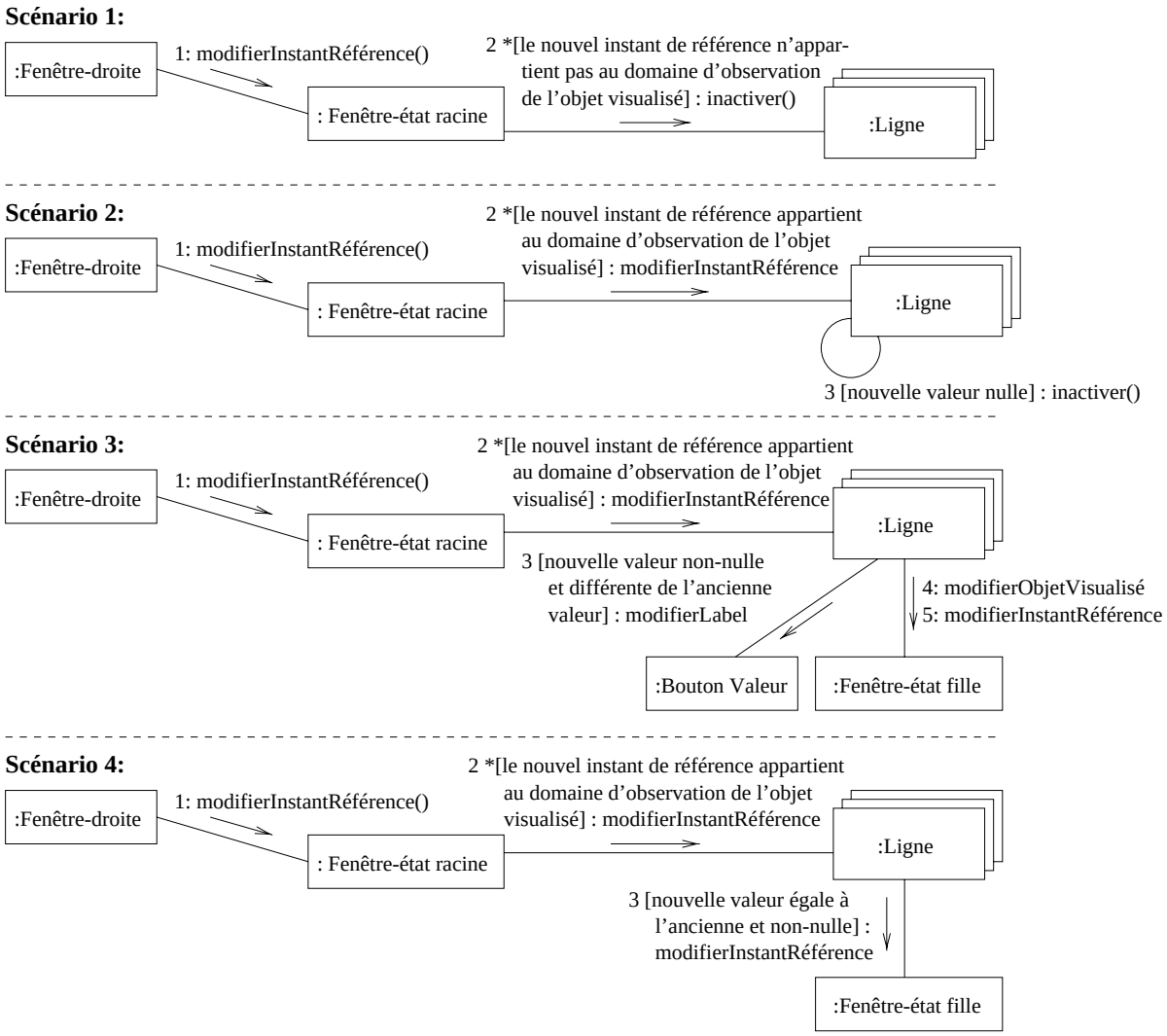



FIG. C.5 – *Diagrammes de collaboration modélisant le traitement au niveau des fenêtres-états d'une modification de l'instant de référence.*