Manipulation de données : un curseur entre la modélisation et l'optimisation de requêtes

Nicolas Travers

Habilitation à
Diriger les
Recherches

présentée par **Nicolas Travers**,
Maître de Conférences au *Conservatoire National des Arts et Métiers*
section CNU n° 27

Manipulation de données : un curseur entre la modélisation et l'optimisation de requêtes

Soutenue le 2 Juillet 2018 devant le jury composé de :

Rapporteurs :

Mme Sihem Amer-Yahia, Directrice de Recherche 1ère classe du CNRS, laboratoire LIG
M. François Goasdoué, Professeur à l'Université de Rennes 1, Laboratoire IRISA
M. Themis Palpanas, Professeur à l'Université de Paris-Descartes, Laboratoire LIPADE

Examinateurs :

Mme Anne Doucet, Professeur à Sorbonne Universités - UPMC, Laboratoire LiP6
M. Pascal Molli, Professeur à l'Université de Nantes, Laboratoire LS2N
M. Philippe Rigaux, Professeur Universitaire au CNAM Paris

Remerciements

Notice :
— *Force* = recherche ;
— *Jedi* = chercheurs ;
— *Maître Jedi* = (HDR | Professeur) ;
— *Chevalier Jedi* = Maître de Conférences ;
— *Padawan* = doctorant.
— *Académie Jedi* = CNAM (laboratoire CÉDRIC et département Informatique).

Je souhaite remercier les membres du *Conseil des Jedi*, que sont les rapporteurs et membres du jury, pour le temps passé à étudier mes travaux, et à apporter leur lumière sur le domaine de la recherche.

Je remercie tout particulièrement les *Maîtres Jedi* qui ont chacun insufflé un tournant majeur dans mon chemin vers le *côté clair de la Force* : Georges Gardarin pour ma formation comme jeune *Padawan*, Michel Scholl pour mes orientations de *Chevalier Jedi*, et Philippe Rigaux dans ses subtils et combien précieux éclaircissements pour prétendre au titre de *Maître Jedi*.

Un clin d'oeil particulier à Cédric du Mouza, avec qui nous formons un duo de *Jedi* de choc, aussi bien dans nos travaux de recherche que durant les pots et les bières. Je te rejoins au rang de *Maître*.

Bien sûr, un grand merci à toute l'équipe Vertigo dans laquelle j'ai pu grandir dans *la Force* : Philippe, Michel C., Marin, Raphaël, mais aussi avec les anciens : Michel S., Dan, Virginie.

Sans oublier les *Padawan* sans qui la *Force* ne serait pas aussi puissante : Quentin, Zeinab, Jordi, Hicheme, et tous les autres avec qui j'ai pu partager de bons moments.

Mais aussi l'*Académie Jedi*, qui contribue au développement de la *Force* et la diffusion de ses connaissances. Je remercie chacun d'entre vous pour tous ces bons moments passés ensemble, je ne citerai personne pour n'en vexer aucun : les enseignants-chercheurs, l'équipe administrative, les ingénieurs...

Et d'un point de vue plus personnel :

À toute ma famille et mes amis qui se sont toujours intéressés à mon travail et qui permettent d'alléger la lourde tâche de *Jedi* qui nous incombe, je vous dis merci.

Michel et Marianne sans qui je n'aurai pas pu travailler sur ce mémoire en toute sérénité ses longues soirées de vacances à Piriac-sur-Mer, ma *Tatooine* à laquelle on revient à chaque épisode.

Hélène pour son amour et son soutien permanent, je sais que tu accepteras (un jour) mes délires sur l'univers de *Star Wars*. Et naturellement, Anaëlle et Lina, sans qui j'aurai sans doute fini cette HDR il y a bien longtemps, mais sans qui elle n'aurait ni valeur ni saveur !
Elles sont toutes les trois la preuve vivante qu'il est possible d'aimer et être *Jedi*.

Si avec toi la force est, le côté obscur ici tu ne verras pas

Chapitre

Table des matières

Chapitre 1

Introduction

Cette thèse résume les travaux de recherche effectués au Cnam, après avoir obtenu ma thèse de doctorat en décembre 2006 [Tra06]. Durant cette période, je me suis principalement focalisé sur trois thématiques relatives au domaine des bases de données, que sont les systèmes de publication/souscription (Pub/Sub), la gestion de larges collections de documents basés sur des artefacts graphiques, et la conception d'une base de données de partitions musicales. Ces thèmes semblent très différents, cependant je les ai abordés avec une approche commune et, j'espère, cohérente. Cette vision globale m'a permis de lier ces types de données hétérogènes, mais également de raffiner et d'approfondir les méthodes élaborées durant mon doctorat avec XML et XQuery. C'est cette approche que j'ai choisi de développer dans cette thèse d'Habilitation à Diriger des Recherches.

Pour revenir brièvement à mes travaux sur XQuery durant ma thèse, j'ai étudié l'optimisation de requêtes dans un contexte distribué avec des données du type XML. Cela a abouti au développement du prototype XLive [TDN07b], un médiateur de données XML, évaluant des requêtes XQuery. Pour ce faire, nous avions défini la *XAlgèbre* qui s'inspire de l'algèbre relationnelle avec quelques opérateurs dédiés. À l'aide d'un modèle de représentation de requêtes, à base de motifs d'arbres ("*Tree Pattern Queries*"), appelé *Tree Graph View* [TDNL07a] (TGV), nous avons pu proposer une simplification du processus d'optimisation de requêtes dans un milieu distribué à l'aide d'un moteur de règles de transformation [TDN07a].

Ces travaux ont continué après ma soutenance de thèse grâce au soutien du projet RNTL *WebContent* (2006-2008) et ont conduit à plusieurs publications scientifiques [AAC+08, TDNL07a, TDN07a, DNT07, TDNL07b, TDN07b]. Je ne résumerai pas ces travaux dans ce mémoire, car l'essence même de ces contributions est présentée dans ma thèse de doctorat. Toutefois, il est intéressant de constater qu'une méthodologie de gestion de la donnée émerge et se conforte dans mes travaux actuels.

Depuis lors, j'ai travaillé sur trois projets de recherche : flux de données textuelles dans le projet RoSeS, base de données d'assets graphique dans le projet Polymathic et interrogation de partition de musiques dans le projet Scorelib. Je présente ici le cadre de mes recherches puis j'introduirai mon analyse sur celles-ci.

1.1 Systèmes de Publication/Souscription : Le projet RoSeS

J'ai tout d'abord travaillé sur le projet RoSeS (ANR-07-MDCO-011, *Really Open and Simple Web Syndication* [1]). Celui-ci a eu pour but de fournir des services et outils pour la syndication du Web, tels que la localisation de sources, l'interrogation, la génération, la composition ou la personnalisation de flux de données.

Cette syndication Web, plus communément connue sous le nom de flux RSS [RSS03], est utilisée sur la plupart des sites Web pour diffuser de l'information à tout utilisateur abonné. Le site Web source produit des informations, appelées *items*, introduits dans un fichier RSS ou Atom. L'abonnement, nommé *souscription*, à ce fichier (par une application) engendre alors un flux d'items entre la source et l'utilisateur, générant des *notifications* et créant ainsi une dynamique entre le site et l'internaute.

Le principal défi à relever dans l'intégration d'un système de Publication/Souscription (Pub/Sub) est la gestion de l'ensemble des souscriptions utilisateurs, et du flux continu d'items générés. En effet, contrairement aux SGBD classiques, les systèmes Pub/Sub reposent sur un paradigme opposé ; les requêtes sont stockées, et les items "interrogent" les requêtes, comparables à des déclencheurs. Dans ce mémoire, je présenterai les choix que nous avons faits pour modéliser ces flux de données, la façon de les interroger et l'optimisation de tels systèmes Pub/Sub.

Pour comprendre en profondeur le comportement des flux RSS, j'ai réalisé une analyse de leur contenu (cf. section 3.1.1) dont les résultats ont permis d'identifier deux types de besoins pour les bases de données : l'exploitation de la structure des flux avec XML et la manipulation de flux orientés textes. Le

1. http://www-bd.lip6.fr/roses

premier permet d'extraire la structure et de la manipuler (cf. section 3.1.2), tandis que le second se focalise sur la composition du contenu textuel et permet ainsi de fournir du sens à l'interrogation (cf. section 3.1.3).

La définition de ces deux modèles de données m'a conduit, de fait, à deux langages d'interrogation. Le premier, le langage RoSeS, exploite la structure et le contenu des items, pour les modifier et les combiner afin de produire de nouveaux flux de données à l'aide d'opérateurs "continus" (cf. section 4.1.1). À l'opposé, le second langage, très simple, repose sur le langage naturel à base de mots-clés comme dans les techniques de recherche d'information traditionnelles [BYRN99] (cf. section 4.1.2). Ce dernier, appelé FiND, permet toutefois la définition formelle d'un modèle de pertinence pour les notifications, basé sur les items déjà délivrés à l'utilisateur final. Le modèle de pertinence repose sur la nouveauté de l'information, ainsi que sur la diversité des informations fournies.

L'optimisation de requêtes se focalise sur la mutualisation des opérateurs communs à l'ensemble des requêtes stockées. Ainsi, pour le langage Pub/Sub RoSeS, j'ai travaillé sur une technique de factorisation et d'optimisation de ressources pour chaque opérateur utilisé lors de l'évaluation des requêtes (cf. section 5.1.1). Dans le cas des souscriptions orientées textes, je me suis focalisé sur le traitement des mots-clés pour des millions de requêtes et l'optimisation du calcul de la pertinence des notifications (cf. section 5.1.3). Ainsi, deux approches distinctes ont pu être implémentées pour le modèle FiND : une approche orientée indexation (cf. section 5.1.2), et une approche orientée distribution dans une base NoSQL (cf. section 5.1.4).

1.2 Bases de données documentaires : Le projet Polymathic

Dans le second projet de recherche, *Polymathic Machine* (FUI 2011-2014), j'ai travaillé sur la définition d'une plateforme de stockage, de partage et de recherche (y compris par le contenu) de larges collections d'artefacts graphiques (*ex.*, images, textures, mouvements). Avec des partenaires industriels issus du monde du jeu vidéo et du traitement d'images (Bulkypix [2], Kyloton [3] et Mocaplab [4]), nous souhaitions définir un système de partage d'assets graphiques produits par les compagnies de jeux vidéo afin d'améliorer les pratiques des artistes et développeurs en automatisant la gestion de l'ensemble de leurs données quel qu'en soit le type.

La principale difficulté pour un tel système est symptomatique des besoins en matière de gestion de bases de données. Le modèle de données doit être capable d'intégrer des images ou ensembles d'images, des métadonnées variées, voire des interconnexions entre les assets. Le langage en lui-même est le reflet des besoins du métier avec des transformations, des recherches par le contenu (images ou métadonnées), une gestion intuitive d'assets et de toutes productions dérivées. Enfin, l'implémentation requiert une intégration de composants multiples avec le traitement d'images, les métadonnées et de nombreuses possibilités de transformations liées au métier, ainsi qu'une intégration simplifiée pour éviter un effort pour les artistes qui l'utiliseront.

Pour répondre à cette problématique, j'ai travaillé sur la définition d'un modèle de données orienté documents (cf. section 3.2), permettant une multitude de représentations possibles pour un même asset. Ce modèle permet ainsi d'intégrer des descripteurs de contenu pour la recherche par similarité (2D, 3D, *motion capture*), des contenus dérivés (*thumbnails*), et des métadonnées variées (propriétaire, artiste, catégories). Un des avantages de la modélisation avec des documents est aussi d'intégrer de la flexibilité dans la définition des assets, favorisant l'évolution du système. Le langage doit alors prendre en compte cette flexibilité avec une structure variable d'un asset à l'autre.

Le langage quant à lui répond au problème de la modularité avec une approche déclarative inspirée de Datalog, que nous avons étendu avec la gestion de fonctions et de données documentaires (cf. section 4.2). Ce langage à base de règles impose un schéma aux collections de données, mais aussi facilite la production de données par inférence. De mon point de vue, le point fort de cette approche est la possibilité de

2. http://bulkypix.com
3. http://www.kylotonngames.com/
4. http://mocaplab.com

rendre cette base de données vivante et de gérer la cohérence des données, simplement en définissant des règles *extensionnelles* pour produire de nouvelles données à l'instar des vues matérialisées, ou des règles *intensionnelles* pour la recherche par le contenu.

Afin de répondre aux problèmes d'intégration d'un tel langage, j'ai pu définir une architecture de stockage et de traitement distribuée avec la base de données NoSQL `MongoDB` [5] (cf. section 5.2). Un système de matérialisation des règles extensionnelles favorise l'intégration du contenu de la base de données et son évaluation. L'architecture distribuée a permis de résoudre plusieurs problématiques, telles que la gestion des documents avec des requêtes multicritères, ou le passage à l'échelle sur la recherche par le contenu, traditionnellement effectué par des index multidimensionnels. De plus, les besoins de l'utilisateur final ont été intégrés grâce à un système de fichiers virtuel pour lequel chaque interaction de l'utilisateur avec son environnement se traduit par une requête `Datalog` sur la base de données, peuplant le contenu de chaque répertoire avec le résultat de la requête.

1.3 Manipulation de corpus musicaux : Le projet Neuma

Dans la continuité du projet ANR Neuma (CONTINT 2008-2011), j'ai pu intégrer le consortium musical Humanum [6] en travaillant sur des extensions de la plateforme Neuma/`Scorelib`. La base documentaire `Scorelib` contient un corpus musical provenant de plusieurs sources dont les partitions sont encodées dans deux formats XML dédiés : MusicXML [Goo01] et MEI [Rol02, MEI17]. Le but de cette base est de fournir un accès libre aux partitions de manière transparente et homogène, produire des statistiques sur le contenu des corpora, proposer des représentations conceptuelles, faire des recherches par le contenu musical, et intégrer des fonctionnalités d'interrogation et de manipulation. C'est sur ce dernier point que je baserai mon analyse, car il présente une perspective intéressante en matière de modèle de données, langage et optimisation.

Pour modéliser un *corpus* musical, il faut comprendre la structure de la notation musicale utilisée par les compositeurs. Des corpora sont un ensemble structuré, composé d'une hiérarchie de corpus (style, compositeur, types d'opus). Tous les opus d'un même corpus respectent un style musical spécifique (quartet, concerto, psaumes à une voix) et contiennent un ensemble de métadonnées décrivant cet opus. La partition musicale elle-même est intégrée à l'opus et décrit le contenu musical, aussi bien les séries de notes que la manière de les représenter. Du point de vue bases de données, cette structure riche est intéressante à exploiter, toutefois complexe à modéliser pour en faciliter l'interrogation. Tout particulièrement la notion de graphisme musical qui permet de mettre en valeur la notation musicale d'une partition, mais qui *in fine* peut être considérée comme une feuille de style et non comme du contenu musical. L'enjeu d'une modélisation épurée de la notation musicale est tel qu'il peut permettre de se focaliser sur le contenu et non la forme.

Créer un modèle de données pour les corpora musicaux est ainsi complexe. J'ai pu travailler sur la conception d'une hiérarchisation des corpora et une modélisation de la notation musicale. Le but était d'abstraire l'ensemble pour se focaliser au maximum sur le contenu de la notation, pour en favoriser les manipulations, interrogations et interactions. La modélisation d'une partition musicale doit également prendre en compte la nature particulière de la structure musicale : les voix, les notes et le temps. D'autant que ces trois notions sont subtilement interconnectées pour former le rendu musical. La notion de séries temporelles s'applique assez naturellement à ces données, en y intégrant des concepts multidimensionnels. Je développerai ce modèle dans la section 3.3.

Au modèle de notation musicale avec des séries temporelles, nous pouvons alors associer, à l'instar du modèle relationnel, une algèbre composée d'opérations de transformation dédiées. La particularité étant les fortes corrélations entre chaque événement composant les séries. Cette algèbre permet ainsi d'extraire des fragments musicaux, de les composer, de les transformer pour produire de nouvelles partitions musicales. J'ai pu alors proposer d'interroger la notation musicale avec le langage XQuery, favorisant la notion de séries auxquelles sont appliquées des fonctions de transformation (section 5.3).

5. `https://mongodb.org`
6. `https://humanum.hypotheses.org/503`

Toutefois, la dissociation du modèle de données de l'encodage réel des partitions de musique implique une transformation du format d'origine vers notre format. Les outils, les encodages existants manipulent très largement les formats MusicXML et MEI, il n'est donc pas envisageable d'imposer un nouveau modèle de sérialisation de la notation musicale. L'implémentation des opérateurs de l'algèbre et du langage sur notre modèle de données doit donc passer par une correspondance de l'encodage vers le modèle (section 5.3).

1.4 Analyse sur mes travaux

Les données et le langage d'interrogation associé forment le pilier sur lequel reposent toute base de données ; la richesse et la puissance du langage s'appuient sur la définition du modèle de données sous-jacent. Toutefois, malgré cette puissance d'expression, la complexité du langage réside également dans la spécificité du modèle de données. De fait, la définition ou le choix du langage d'interrogation sont intrinsèquement liés à ce modèle de données qui va permettre d'en exploiter les spécificités et de produire un langage adapté aux besoins de l'utilisateur dans le contexte lié aux données. Du fait de cette richesse et cette complexité, le domaine des bases de données est en continuelle évolution : de nouveaux types de données, de nouveaux langages d'interrogation, de nouvelles techniques d'optimisation.

Ainsi, cette thèse d'HDR a pour but de mettre en valeur une approche méthodologique, au-delà des domaines spécifiques auxquels elle s'applique. Mes travaux ayant été basés sur des projets de recherche (`RoSeS`, `Polymathic` et `Scorelib`) avec des données bien différentes et des besoins d'interrogation bien distincts, il eut été difficile de comparer les travaux en découpant ce mémoire par projet. L'approche serait restée trop générale. Ainsi, je présenterai mes travaux de manière transversale en y décrivant les trois étapes de cette méthodologie :

(1) La **modélisation** : flux de documents XML, flux de textes, documents multi structurés, séries temporelles structurées. La modélisation des données intégrées au système est le socle sur lequel repose l'ensemble du processus. Il permet en effet de définir les manipulations possibles et les contraintes inhérentes aux types de données ;

(2) Le **langage** : Le langage `RoSeS`, des souscriptions de mots-clés, des règles `Datalog`, XQuery. Le langage permet à l'utilisateur final d'avoir une puissance d'expression sur la gestion des données, tout en gardant un haut niveau d'abstraction sur la manière de l'obtenir. Cela permet à l'utilisateur de se dégager des contraintes techniques liées à l'intégration et l'implémentation du langage ;

(3) Le **système** : Optimisation multi requêtes, évaluation de flux de textes par indexation ou distribution en NoSQL, évaluation de requêtes XQuery sur séries temporelles virtualisées. Reposant à la fois sur le modèle et le langage, cette partie prend alors tout son sens. En effet, la combinaison des deux étapes précédentes donne lieu à des choix d'implémentation indispensables pour le bon fonctionnement du système. Cela passe par des besoins d'indexation, de distribution, d'optimisation de graphes ou de ressources.

Finalement, ces trois points caractérisent les étapes principales de la gestion de données. Mon mémoire va donc se découper en trois étapes qui vont détailler les choix réalisés pour chaque type de données et en quoi ils impactent les étapes suivantes. Cela montrera par exemple dans quelle mesure les divergences de modèles de données ont des implications aussi bien sur le langage que sur l'implémentation, et comment, pour un même langage, plusieurs implémentations sont également possibles.

1.5 Données factuelles

Résumé des publications sur la période

— **3 articles dans des journaux internationaux** (Inf. Sys., JCST, IJWIS [7]) ;
— **20 articles dans des conférences internationales** (VLDB, EDBT, SSDBM, CIKM, WISE, K-CAP...) ;
— **3 chapitres de livre**, dont 1 en cours de relecture ;

7. *Best Paper Awards 2014*

— **2 articles dans des journaux nationaux** (ISI) ;
— **10 articles dans une conférence nationale** (BDA, JIM) ;
— **1 ouvrage en ligne (OpenClassrooms)** ;
— Titulaire de la PEDR en 2017.

Prototypes

— Le système `RoSeS` a été développé par Jordi Creuse Tomàs durant sa thèse. J'ai participé à la définition du modèle de données, le langage, l'optimisation et l'architecture (section 5.1.1).
— Le système `FiND` a été implémenté par Zeinab Hmedeh, Sylvain Bouquin et moi-même. Deux implémentations ont été réalisées, la première centralisée (section 5.1.2), et la seconde en milieu NoSQL 5.1.3).
— La plateforme `Polymathic` a été développée par Emilian Pascalau (Ingénieur de Recherche) pour la partie serveur Web (section 5.2.1), et par moi-même pour la partie "bureau interactif" avec FUSE (section 5.2.2). La conception du système a été faite par Philippe Rigaux et moi-même.
— La plateforme `Scorelib` a été réalisée initialement dans le projet ANR Neuma puis le projet ANR MuNiR. Une seconde version a été réalisée en *Django* dans l'équipe Vertigo par Raphaël Fournier-S'niehotta, Philippe Rigaux et moi-même. J'ai conçu le moteur de requêtes ScoreQL (section 5.3).

Encadrements

Thèses de doctorat

— "*Indexation pour la recherche par le contenu textuel de flux RSS*" par Zeinab Hmedeh (Cedric, CNAM), le 13/12/2013. Co-encadré avec Michel Scholl jusqu'en 2011 et Cedric du Mouza.
— "*RoSeS : Un moteur de requêtes continues pour l'aggrégation de flux RSS à large échelle*" par Jordi Creus Tomàs (Lip6, UPMC), le 10/12/12. Encadré par Bernd Amann et Dan Vodislav. Durant sa thèse, j'ai contribué à la définition du modèle de données, le langage, l'optimiseur et le système.
— "*A Real-time recommandation system for micro-blogging*" par Quentin Grossetti. En thèse depuis septembre 2014, encadrée par Cedric du Mouza, Camélia Constantin et moi-même. Je traiterai de ces travaux en cours dans la conclusion.
— "*La réconciliation des bases orientées colonnes et orientées lignes avec T-Plotter*" par Hichem Chaalal. Thèse en cours depuis 2011 à l'université d'Oran en Algérie (USTO), encadrée par Belbachir Hafida. J'ai commencé à co-encadrer son travail depuis Mai 2016.

Stages de Master 2 - parcours recherche

— "*Indexation par listes inverses pour les flux RSS*" par Zeinab Hmedeh. Co-encadrée avec Michel Scholl et Cédric du Mouza (Septembre 2010). Ces travaux préliminaires du projet RoSeS ont permis de définir le modèle de données orienté texte et une première solution d'indexation Pub/Sub sur des flux textuels (index CIL de la section 5.1.2).
— "*An Intelligent Publish/Subscribe System at Web Scale*" par Han Miyoung. Co-encadrée avec Cédric du Mouza (Septembre 2014). Ce stage avait pour but de proposer une implémentation alternative du système `FiND` (section 5.1.4) dans un environnement NoSQL (`MongoDB`). Elle est actuellement en thèse avec Pierre Senelart et Stéphane Bressan à Telecom Paristech et l'Institut Mines-Télécom.
— "*Efficient TDV computation in NoSQL environment*" par Ouassim Lalaoui. Co-encadré avec Cédric du Mouza (Septembre 2015). Ce travail s'est focalisé sur la mise à jour des poids des termes "TDV" dont nous parlerons dans la section 3.1.3.
— "*Détection et éclatement des bulles d'information dans les systèmes de recommandation*" par Simin Gao. Co-encadrée avec Cédric du Mouza et Quentin Grossetti (Septembre 2017). Durant ce stage, elle a pu compléter les travaux de thèse de Quentin Grossetti sur la détection de communautés thématiques sur un réseau Twitter et le moyen de réduire les "effets bulles" des retweets.

— *"requêtes complexes dans un corpus de hiéroglyphes"* par Jules Vandeputte. Co-encadré avec Serge Rosmorduc (Septembre 2017). La nature des traductions des hiéroglyphes et l'utilisation de corpus par les égyptologues ont donné lieu durant ce stage à la création d'une moteur de recherche d'information sur des données corrélées (hiéroglyphe, transcription, translitération et traduction).

Mémoires d'ingénieur

Durant la période, j'ai également encadré 20 mémoires d'ingénieurs CNAM (~9 mois). Parmi ceux-ci, deux sujets sont liés directement à mes travaux de recherche :

— *"The `Polymathic` machine"* par Emilian Pascalau. Intégration de `Polymathic` avec `MongoDB`.

— *"Mise en œuvre d'un Système de Publication/Souscription basé sur les Flux d'Information de type RSS"* par Sylvain Bouquin. Il a effectué un travail de ré-ingénierie du système `FiND` pour en faire une version exploitable à large échelle avec MongoDB.

1.6 Plan du mémoire

Ce mémoire suivra donc ce fil directeur "la manipulation de données" en conservant ce découpage : modélisation, interrogation, optimisation.

Le **chapitre 2 (État de l'art)** présentera les bases de littérature du traitement de données dans les trois domaines :

— Les *Systèmes de gestion de flux de données* en s'intéressant particulièrement au traitement de requêtes continues, l'optimisation multi requêtes, la problématique de l'indexation dans le Pub/Sub et le filtrage par nouveauté et diversité.

— Les *bases de données documentaires* avec un accent sur les systèmes basés sur des règles, ainsi que les systèmes de fichiers virtuels.

— La *manipulation de corpus musicaux* avec les approches orientées langages d'interrogation, mais également la représentation des partitions de musique.

Dans le **chapitre 3 (Modélisation de données)**, je présenterai la problématique de modélisation de données et les choix que j'ai pu proposer pour résoudre ces problèmes. Une étude sur le contexte permet d'orienter les décisions et de mieux comprendre par la suite les enjeux, ainsi que les besoins de manipulation. Je détaillerai les deux orientations de modélisation pour les flux RSS (XML vs texte), la représentation d'assets graphiques dans un modèle semi-structuré et la manière d'abstraire la notion de contenu musical avec des séries temporelles à partir de partitions de musique.

Le **chapitre 4 (Manipulation et interrogation de données)** repose sur les modèles de données proposés afin de présenter pour chacun un langage d'interrogation adapté. Je comparerai les différentes formalisations des opérations ou règles de manipulation, je mettrai également en perspective les différents langages pour en expliquer les subtilités (flux XML vs séries temporelles étendues aux objets, datalog vs requêtes textuelles). La motivation de ces choix se base sur les possibilités d'interrogation favorisant la manipulation de données.

Le **chapitre 5 (Intégration et Optimisation de requêtes)** détaillera les choix d'implémentation de chaque approche, je discuterai des problématiques liées aux choix effectués, mais également la manière d'intégrer le langage et les opérations dans un tel système. Je présenterai les variantes possibles d'implémentation d'un même langage, avec particulièrement la comparaison entre une implémentation centralisée et distribuée dans le cadre du Pub/Sub.

Pour finir, le **chapitre 6 (Conclusion et perspectives)** résumera l'ensemble de l'approche abordée dans ces différentes parties pour apporter une vision globale et une réflexion sur les choix cruciaux en matière de manipulation de données. Des travaux en cours et en perspectives seront ensuite présentés avec le prisme de mon approche afin de mieux l'illustrer.

Chapitre 2

État de l'art

Je vais développer dans ce chapitre les notions qui ont inspiré mes travaux sur l'interrogation de données. Le contexte associé à chaque type de données permet de mieux comprendre les tenants et aboutissants du domaine, et ainsi de mieux adapter le modèle de données aux problématiques de manipulation, le langage aux usages métiers et enfin l'implémentation sur ces deux critères et leurs contraintes techniques. Dans cette cartographie sur la manipulation de données, je me positionnerai d'un point de vue du "traitement de données" pour en faciliter la compréhension. Les différentes notions présentées sont les systèmes de gestion de flux de données ou *Data Streams Management Systems (DSMS)*, le langage Datalog et les bases documentaires, puis la gestion de séries temporelles dans le contexte des partitions musicales.

2.1 Systèmes de gestion de flux de données

Cette section présente différentes approches permettant la manipulation de flux de données. Comme nous pourrons le constater, ces systèmes appelés *Data Stream Management Systems* (DSMS) s'intéressent principalement au filtrage de données en continu. Pour positionner mes travaux et comparer les deux approches détaillées par la suite, j'ai choisi de ne présenter que les systèmes s'attaquant au problème de traitement de données structurées (plus particulièrement semi-structurées avec XML - RSS & Atom), ainsi que les approches basées sur les flux textuels avec des mots-clés.

2.1.1 Requêtes continues sur données structurées

Traitement de flux de données. La manipulation de données en continu repose sur une problématique inverse à celle des bases de données traditionnelles. En effet, les données arrivent en continu et les requêtes sont persistantes. De fait, le contenu de la base de données est en constante évolution et de nombreuses requêtes doivent être évaluées en temps réel. Ainsi, les principales propositions dans le domaine des DSMS s'orientent sur un modèle de flux de données combiné à une algèbre "continue" traitant ce flux continu [GÖ03, LTWZ05, WDR06]. La particularité de ces opérateurs continus est de définir des fenêtres temporelles [LMT+05] sur le flux (ou snapshot), les opérateurs classiques pouvant alors s'appliquer naturellement sur le contenu de cette fenêtre. Afin de faciliter la manipulation de ces flux, des langages de requêtes continus ont été proposés, nous pouvons particulièrement citer CQL [ABW06], AQuery [LS03], NiagaraCQ [CDTW00] ou une extension pour XQuery [BFF+07].

Une fenêtre appliquée à un flux de données permet de produire une vue temporaire sur les données. Ainsi, les opérateurs relationnels peuvent manipuler ces vues en utilisant le pipeline et des piles. Plusieurs types de fenêtres ont été introduits dans la littérature [PS06], le contenu d'une fenêtre à un instant t est défini par deux propriétés fondamentales : le déclenchement et les bornes. Le déclenchement de l'ouverture ou la fermeture d'une fenêtre se base sur le type d'événement : sur le temps (*time-based windows*) ou sur l'activité des items (*item-based windows*). Ainsi, le déclenchement du calcul de cette fenêtre dépend de la contrainte associée au temps ou à l'item. Quant au contenu de la fenêtre, il est déterminé par le type de bornes utilisé. Trois types de bornes orientent le contenu avec des fenêtres de type : *sliding* (FIFO), *tumbling* (un item n'appartient qu'à une seule fenêtre) et *landmark* (accumulation de contenu).

Les combinaisons de déclencheurs et de bornes sont montrées dans la figure 2.1. On peut d'un côté constater le rythme de changement des fenêtres (le rythme temporel est différent des items), et le contenu qui peut être très différent en fonction des bornes. Chaque combinaison porte un intérêt particulier dépendant des besoins d'interrogation, et de fait applicatif ; on peut vouloir avantager la place mémoire en s'orientant vers de l'*item-based*, prendre en compte les fréquences de calcul (dépend du débit du flux), faire des moyennes globales (*landmark*) ou locales (*tumbling*), étudier les variations en continu avec du *sliding-window*... Les systèmes Pub/Sub se basent principalement sur des fenêtres glissantes basées sur le temps ou les items, communément dénommées *time-windows* et *count-windows*.

L'évaluation de flux XML, comme RSS et Atom, a été proposée avec des extensions pour XPath [GS03,

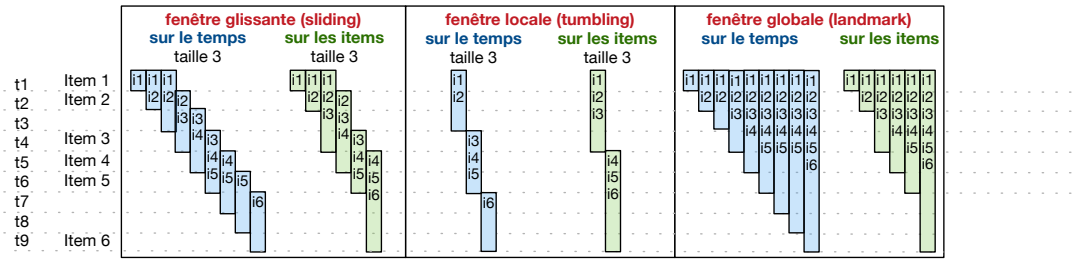


FIGURE 2.1 – Différents types de fenêtres : sliding/tumbling/landmark et temps/item

PC03] et XQuery [KSSS04, BFF+07]. Toutefois, ajouter une sur couche à des langages dédiés aux bases de données XML statiques augmente la complexité d'expression et d'implémentation. Je propose dans ce mémoire un langage dédié aux besoins que sont ici des flux RSS contenant des fragments XML simples. Le langage doit à la fois être expressif et s'intégrer efficacement dans un système Pub/Sub.

Agrégation de flux RSS. De nombreuses applications proposent le traitement et la manipulation de flux RSS. Ces outils vont de simples extensions de navigateur pour souscrire à un flux jusqu'à la consultation de flux avec des possibilités d'agrégations riches (c.-à-d., *Google News*). Ces derniers proposent de personnaliser les flux RSS par filtrage ou regroupement de différentes sources de données. Toutefois, cette agrégation est très souvent limitée à la création de simples *widgets* (mini application Web), comme *Netvibes* ou *Feedzilla*, voire capables de créer des collections de flux *Google Reader* (abandonné le 1er janvier 2013). Pour *Google Reader*, nous pourrons constater que ce système permettait de créer des hiérarchies de collections de flux RSS que l'on peut interroger avec des mots-clés, mais de manière statique.

À l'opposé de ces outils graphiques simples, nous nous intéressons aux systèmes de filtrages basés sur le contenu capables d'intégrer des manipulations plus riches de filtrage et d'agrégations tout en gardant le concept de traitement de flux de données en continu.

YahooPipes ![Yah07] est sans doute l'approche répondant le plus à notre problématique. Il propose une approche d'agrégation pour RSS sous forme de *widgets*. Il propose de créer des "*pipes*", des opérations représentées graphiquement grâce au langage *Yahoo ! Query Language* (YQL) [Yah08]. Ce langage est fortement inspiré par le langage SQL permettant la création de tables d'agrégation par des extractions de sources Web (dont RSS). Ainsi, le traitement des données par YQL est basé sur un modèle statique avec un stockage des extractions pour des interrogations régulières (fenêtre *landmark* sans déclencheurs). Malheureusement, comme nous le verrons par la suite, une telle représentation ne favorise pas les optimisations multi requêtes (section 4.1.1) et passe difficilement à l'échelle pour de nombreuses requêtes.

Optimisation multi requêtes. Ce problème de passage à l'échelle est d'ailleurs un problème crucial dans le cadre du Pub/Sub. En effet, le nombre de souscriptions utilisateurs est tel qu'il est nécessaire de mutualiser les traitements pour minimiser la consommation de ressources. Si plusieurs requêtes continues s'appliquent sur un unique flux avec des opérations en commun, cela devient comparable aux problèmes d'optimisations multi requêtes en relationnel (MQO [Sel88]). Ces techniques tentent de maximiser la factorisation des traitements communs [DGH+06]. Nous pouvons trouver des solutions basées sur l'indexation de prédicats [WGMB+09], partages d'états dans des automates [DGH+06], graphes de jointures [HDG+07] et factorisation de sous-requêtes [CDTW00, ABW06, CCD+03].

Ces approches ont inspiré notre solution en cherchant à mutualiser les traitements en se basant sur un graphe multi requêtes. Le contenu des items du flux devient alors un facteur majeur de filtrage et de factorisation pour les systèmes Pub/Sub. Nous pourrions également citer certaines techniques qui se concentrent sur les thèmes (topic-clustering) [LYD+07, MZV07], structures d'indexation de souscriptions adaptées [KCM09, FJL+01, CPY07], et traitement de flux distribués [RMP+07, JA06]. Malheureusement, ces techniques se focalisent uniquement sur la parallélisation de requêtes filtrantes sans s'attaquer à l'optimisation de l'agrégation de flux qui est partie intégrante du langage Pub/Sub.

2.1.2 Indexation de flux par les mots-clés

À contrario d'un langage Pub/Sub expressif, d'autres approches s'intéressent au traitement du contenu textuel. Le modèle et le langage d'interrogation sont assez intuitifs puisqu'ils reposent sur les approches traditionnelles en Recherche d'Information (RI), toutefois la complexité repose sur l'optimisation de la masse d'information à traiter en temps réel sur des requêtes basées sur des mots.

Ces approches s'orientent vers l'indexation des souscriptions pour filtrer en continu les publications. Elles proposent majoritairement des solutions de comptage"*Count-based*" (CI) plutôt qu'arborescentes "*Tree Based*"(TI). `LeSubscribe` [PFL+00] fusionne des listes de souscriptions pour chaque prédicat utilisé, puis un *comptage* des prédicats pour chaque souscription permet de vérifier si elles sont validées. Le test des souscriptions partiellement validées augmente la quantité de traitements inutiles. Une solution [FJL+01] est de grouper les souscriptions par taille de souscription.

Le système SIFT [YGM94, YGM99] propose une comparaison de trois techniques d'indexation de documents orientés textes. Des implémentations sur le disque des indexes CI avec des listes inverses (IL), un index basé sur le tri des mots (*Ranked key Inverted List* - RIL) avec les listes de souscriptions dont le mot ayant le rang le plus bas (poids du mot très faible) devient le facteur discriminant, et une implémentation orientée arborescente avec tri de mots. Toutefois, cette implémentation reposant sur le disque passe difficilement à l'échelle en matière d'accès permanent aux structures de l'index dû au contexte Pub/Sub. En effet, le problème de la dimension n'est pas comparable (cf. section 3.1.1) avec (a) des items avec 52 mots en moyennes (12K pour [YGM99]), (b) un vocabulaire de souscriptions de 1,5M de mots vs 18K.

Il est intéressant également de comparer les méthodes orientées listes inverses [KCM09] dont le but est de favoriser les notifications de publicités"*bids*" en temps réel. La particularité de l'approche est d'indexer en mémoire les combinaisons de mots-clés les plus fréquentes. Ainsi, l'espace de recherche de souscriptions potentielles est réduit à un sous-ensemble de bids qui sont alors organisés par paquets de combinaisons fréquentes. Très approprié aux recherches sur des moteurs de recherche (2 à 3 mots en moyenne [BJC+04]), le nombre de combinaisons est ainsi restreint et favorise le passage à l'échelle dans ce contexte. Toutefois, dans le cadre qui nous intéresse, l'espace de recherche reste malgré tout ingérable et chaque publication génère une quantité de recherches exponentielle.

Shraer et al. [SGFJ13] s'intéressent à la notification de tweets corrélés à des articles de presse (souscriptions). Ainsi, chaque tweet génère un score correspondant à sa similarité avec l'article associé, si ce score dépasse celui des tweets précédemment notifiés, il est alors notifié. Cette approche *top-k* repose sur une structure d'indexation avec les listes inverses d'articles pour chaque mot qui les compose. Bien sûr, cette solution n'est pas applicable à des millions de souscriptions et des items de plus grande taille.

Ces approches ont influencé les travaux que je présente dans la section 5.1.2, mais au regard du volume de données avec des souscriptions (centaines de millions), des publications ayant des tailles conséquentes (50 à 500 mots), un vocabulaire provenant du Web et une fréquence de plus en plus forte, les solutions apportées restent limitées, et je présenterai les méthodes qui ont favorisé ce passage à l'échelle.

2.1.3 Nouveauté et Diversité

Les utilisateurs recevant les notifications produites par un système Pub/Sub peuvent être submergés par la quantité d'informations "pertinente". Pour améliorer la qualité des notifications, un filtrage par *nouveauté* et *diversité* est nécessaire. Ces deux propriétés complémentaires mesurent le degré de répétition dans les publications déjà reçues par le passé. Le filtrage par nouveauté supprime toute information redondante présente dans un item précédent, tandis que le filtrage par diversité capture la redondance dans l'ensemble des items. Je présente, ici, les approches de filtrages, tout d'abord sur des données statiques, puis dans un contexte Pub/Sub.

Pour effectuer une recherche de documents sur le Web, il faut produire les k documents les plus pertinents à un utilisateur donné tout en gardant une diversité dans les documents retournés. Pour ce faire, on peut trouver des solutions basées soit sur des modèles de diversité probabilistes, [AK11] soit sur des graphes permettant de définir la distance entre les documents [DP12a]. La notion de mesure de diversité est très variable, on peut trouver des modèles basés sur les attributs disjoints entre les items et les

requêtes [YLAY09], sur un mixe entre la similarité et la diversité [SM01], sur des analyses de sentiments avec des algorithmes `Max-Min` [AAYIM13], sur la distance temporelle des items avec une similarité gaussienne [KCC12], voire même en comparant les items grâce à une distance NCD (compression normalisée) [CRYT10]. Plus généralement, ces techniques reposants sur le postulat que des données statiques, favorisent la notion de similarité en triant les données naturellement. Toutefois, dans un contexte Pub/-Sub, la notion de pertinence de l'information et de sa redondance varie au cours du temps et n'est pas compatible avec ces approches. Par ailleurs, le concept de production en temps réel des items ajoute une contrainte extrêmement forte sur la notion de similarité qui devient, de fait, asymétrique.

Les approches Pub/Sub de filtrage continu sont alors combinées avec des techniques de type *top-k*. En utilisant des fenêtres *item-based* sur le flux de données, [DP12b] garantit le passage à l'échelle tout en utilisant un index mis à jour dynamiquement capable de calculer la diversité d'un item sur chaque *snaphot*. Tandis que [MSN11] présente une approche incrémentale de diversification du contenu provenant de fenêtres temporelles. D'autres travaux s'attaquent au filtrage par nouveauté par extraction d'entités nommées [GDH04] ou de thèmes [PDSAT12] avec des distances de couverture thématique. Malheureusement, ces solutions n'exploitent pas la complémentarité de la diversité avec la nouveauté, qui plus est sur des fenêtres disjointes ne favorisant pas la pertinence avec des calculs indépendants sur le temps. D'autre part, les fenêtres *item-based* garantissent l'espace mémoire utilisé, mais à surtout un impact sur le contenu. En effet, un flux lent risque de trop filtrer (filtrage identique avec des items persistants), et les flux rapides ne filtreront pas assez (les items pertinents disparaissent trop rapidement).

La solution proposée par Drosou et al. [DSP09] intègre le filtrage *top-k* en temps réel sur des fenêtres temporelles. La diversité est calculée grâce à un algorithme de maximisation des permutations entre les items du top-k avec le contenu de la fenêtre (ayant du nouveau contenu). La fenêtre considérée prend en compte l'ensemble des items d'une fenêtre temporelle et tente de maximiser en continu le *top-k* en choisissant le meilleur item favorisant la diversité localement. Toutefois, l'aspect temporel peut favoriser l'apparition d'un ancien item ayant été rejeté précédemment (il reste malgré tout dans la fenêtre), ce qui est contraire au principe de filtrage continu préservant la diversité de l'information. Je présenterai une comparaison de cette approche avec la nôtre (Section 5.1.3) car elles sont très similaires sur le principe, mais techniquement très différentes du fait de la notion de permutation. Les expériences permettront de montrer que cette approche maximise localement la diversité, mais pas sur le temps.

2.2 Bases de données documentaires

La gestion de bases documentaires (DMS - *Document Management Systems*) s'intéresse principalement à l'extraction de contenu pour produire des résultats transformés ou agrégés. L'approche que je présente dans ce mémoire propose de traiter ce problème avec un système à base de règles d'inférences, rendant cette base de données active plutôt que procédurale.

2.2.1 *Document Management Systems* et règles d'inférences

Les systèmes de gestion documentaire de références, tels qu'Alfresco One [1], Documentum [2] ou OpenText [3] proposent de gérer des rapports, feuilles de calcul et présentations dans un espace partagé, mais sans exploiter le contenu ou la structure des documents d'un point de vue bases de données (interrogation ou agrégation). Le modèle de données est alors très simple avec des métadonnées et des références, permettant de faire des requêtes assimilées à un moteur de recherche. L'intégration d'un tel système se focalise sur le partage et la collaboration de documents. Le système `Polymathic` propose d'intégrer les informations de manière structurée et de favoriser son exploitation en proposant des règles de transformation pouvant se focaliser aussi bien sur les métadonnées que sur le contenu.

Les travaux [WW10, SN03] s'intéressent aux *workflows* d'intégration de documents comme pour Documentum et Alfresco. Mais la notion de *workflow* définit un processus pour chaque besoin de l'application,

1. `https://www.alfresco.com`
2. `https://www.dellemc.com/en-us/index.htm`
3. `https://www.opentext.com`

rendant le système rigide et difficile à faire évoluer. Une approche basée sur les règles permet de formaliser et pérenniser les transformations dans la base de données et non dans le système d'information, il favorise ainsi les dérivations de contenu, la composition, la recherche de contenu et l'interopérabilité.

Une base de données reposant sur des règles rend la base de données "*Active*" de manière analogue aux *Triggers* [WC96, BCP98]. Quel que soit le modèle de données sous-jacent, la spécificité des règles réside dans le langage de définition de celles-ci. Loin du standard SQL, un langage actif basé sur des règles comme `Datalog` [AV91] permet de déclencher les règles pour chaque donnée extensionnelle (faits) pour générer des données intensionnelles (déductions). Chaque mise à jour génère donc des données intensionnelles pouvant elles-mêmes générer de nouvelles données par répercussion. Ce langage permet de spécifier des règles favorisant l'enrichissement de données et la recherche de données. Cette combinaison active/déductive a été présentée dans `Statelog` [LLM98], avec quelques variantes d'applications de `Datalog` [dMGFS11, Hel10] dans un contexte distribué (surveillance, alerte, statistique, etc.). D'une certaine manière, les travaux sur `Active XML` [ABM08] et le projet `WebdamLog` [ABGA11] ont influencé nos travaux, tout en restant dans un système cadré pour éviter l'ensemble des problèmes liés aux règles actives complexes (*ex.*, négation, fonctions non déterministes) appelé "*Positive Datalog*".

Les bases de données actives génèrent de nouvelles données à partir des règles, pouvant également déclencher d'autres règles. La convergence et la terminaison des règles sont garanties dans le cadre du *Positive Datalog* [AHW95, BDR04]. Toutefois, le cadre d'application de `Polymathic` impose l'intégration de fonctions au langage impliquant le problème de *Turing-complétude* [AHV95] et de fait rend la terminaison des règles non décidable. D'un point de vue fonctionnel, cette propriété peut avoir de nombreuses répercussions sur la viabilité du système de règles. Toutefois, en pratique, la terminaison des règles peut être contrôlée tout en gardant la puissance d'expressivité du langage avec des fonctions.

2.2.2 Intégration de la base de données dans l'espace de travail

Un des buts de `Polymathic` est d'intégrer l'application dans l'espace de travail de l'utilisateur sans changer ses habitudes. La solution que nous avons envisagée est d'intégrer les interactions avec la base de données active directement dans l'espace de travail. Grâce à un système de fichier virtuel (VFS), le système d'exploitation interroge différentes vues sur la base `Polymathic`. Cette approche VFS est comparable aux systèmes SFS [GJSO91] et LISFS [PR03], dont le contenu des répertoires correspond au résultat de requêtes basées sur le chemin de l'arborescence de fichier. Contrairement à ces approches n'interagissant qu'avec la vue "fichier", le langage `Polymathic` favorise la définition de multiples vues permettant la généralisation du principe en fournissant une multitude d'interactions et de facettes sur les assets graphiques.

Proposer des collections de fichiers *intensionnels* est semblable à la recherche de fichiers comme pour Damasc [BMPT09] ou QUASAR [AMM08]. Mais contrairement au fait de reposer sur une arborescence de fichiers déjà existante pour définir le contenu de la base de données pour les recherches, nous proposons un système de fichier virtuel dont le contenu provient des données stockées et de leurs dérivés.

Je me dois de citer les systèmes de gestion de fichiers sur le *Cloud*, comme Microsoft OneDrive For Business [One17], Dropbox [Dro17], Google Drive [Goo14] ou Amazon s3fs [Ama17], capables de consulter localement le contenu d'un fichier stocké à distance. Toutefois, ce contenu reste une copie statique dans le système de fichier. Notre solution est dynamique permettant à l'utilisateur d'avoir une vue sur la base de données reflétant les règles du système. La fonctionnalité de recherche est de fait plus adaptée, car elle repose sur les besoins métiers plutôt que sur la gestion de fichiers dans le *Cloud*.

2.3 Manipulation de corpus musicaux

La notation musicale est une structure complexe difficile à modéliser et manipuler. C'est un subtil mélange entre la représentation et l'encodage des notes, liant différents instruments, hiérarchies de corpus dont le résultat produit une partition harmonieuse. Toutefois, comment s'attaquer à la gestion d'une telle structure et surtout d'un corpus musical ? Les premières solutions allant dans ce sens reposent sur le principe de Recherche d'Information (*Music Information Retrieval*) avec des recherches non structurées

ou par similarité [TWV05]. Toutefois, ils n'exploitent pas la structure de la notation musicale, et de fait restent de haut niveau sur l'exploitation du contenu musicale. C'est ce qui va nous intéresser dans cette partie.

2.3.1 Interrogation de partitions musicales

D'un point de vue bases de données, il est nécessaire de définir un modèle de données, associé à des opérateurs capables de transformer ces données qui sont ensuite intégrées à un langage de haut niveau pour en faciliter la manipulation avec une bonne expressivité. Certains langages ont été proposés pour la représentation et la manipulation musicale, notamment Haskell pour le système *Euterpea* [Hud15], et bien d'autres [Bal96, JBDC+13, FLOB13]. Mais l'utilisation de tels langages s'oriente sur la génération et non l'interrogation de partition, ne facilitant pas la manipulation de collections de partitions.

Une modélisation à base de séries temporelles est abordée par Baazizi et al., [BBC11] qui exploite la structure XML des encodages musicaux en utilisant les propriétés de *XQuery updates* pour la synchronisation. La génération de séries temporelles, à partir d'automates temporels, proposée par [FBF13] est analogue à l'approche précédente. Toutefois, ces opérations reposent plus sur chaque élément de la série temporelle plutôt que la série elle-même. Il en résulte une forte complexité dans l'expressivité des opérations de manipulation qui impacte l'expressivité et l'utilisation du langage. De fait, l'exploitation d'une partition musicale n'est pas triviale et très dépendante de son encodage.

Les langages de générations de musique ont inspiré nos travaux, tels que Haskell [Hud15] ; langage de programmation fonctionnelle intégrant des types abstraits pour les concepts musicaux. Également T-Calculus [JBDC+13] capture le flux musical avec des opérations d'extraction (*tiled stream*) et de synchronisation (*tiled group*). D'autres langages se focalisent sur les partitions interactives [FLOB13, FOL12] ou des grammaires temporelles [QH13].

Bien sûr, beaucoup de concepts musicaux ont déjà été modélisés (évènements, sons et opérateurs). L'approche que je présente dans ce mémoire se distingue des autres dans le fait qu'il s'inscrit dans une problématique uniquement "bases de données" ; une algèbre finie qui manipule les instances du modèle sous-jacent. L'algèbre définie sous forme close avec une forte expressivité permet d'exprimer de nombreuses manipulations sur l'encodage musical, tout en restant accessible. Ce langage est donc complémentaire des approches précédentes puisqu'il aborde le problème de l'interrogation et non de la génération.

2.3.2 Manipulation de notation musicale

La manipulation des couches de bas niveau pour l'encodage musical est également une tâche qu'il faut intégrer pour l'implémentation d'un tel langage. Humdrum [Hur02] manipule des fichiers pleins texte (ASCII), alors que `music21` [CA10] sérialise le flux musical dans des canaux MIDI organisés en couches. L'importation des formats standards tels que MusicXML ou MEI permet d'utiliser ces outils puissants.

Malgré tout, ces outils sont dédiés à la programmation de scripts pour la manipulation, et peu compatible avec une approche algébrique sur la structure. Toutefois, je présenterai (section 5.3) notre implémentation d'opérateurs algébrique en produisant pour `music21` les implémentations des opérateurs. Chacun de ces opérateurs est appelé par la requête, et crée une chaîne de manipulation sans avoir à créer un programme. Comme nous le verrons, cette approche nécessite quelques optimisations pour pouvoir gérer de gros corpus musicaux.

De l'autre côté, le langage d'interrogation doit permettre de faciliter l'expressivité de l'interrogation. Du fait des standards musicaux en XML, notre choix s'est orienté sur XQuery [XQu07]. Des travaux antérieurs proposent de manipuler des partitions encodées en MusicXML [GSD08], et THoTH [Whe11] en utilisant des motifs. Mais une approche trop proche de l'encodage empêche toute abstraction du modèle musicale et ne facilite pas la manipulation, produisant des programmes dédiés et non génériques.

L'approche que j'aborde dans ce travail s'inspire des travaux de recherche sur les systèmes de médiation pour XQuery [GMUW00, DHI12, AAC+08, TDNL07a] que j'ai pu traiter durant ma thèse. La différence majeure est la gestion des instances qui mêlent aussi bien des données physiques et virtuelles. Cela rappelle le concept d'ActiveXML [ABM08] et de déclencheurs activés à la demande.

Chapitre 3

Modélisation de données

La définition du modèle de données est une tâche indispensable à la conception de toutes bases de données. C'est le socle sur lequel repose la complexité des manipulations des données, dépendant principalement du type de données considéré et de la manière de les représenter. Nous pouvons citer des modèles de bases de données traditionnelles : relationnelles [AHV95], déductives (Datalog [CGT89]), temporelles [BD91], spatiales [RSV01], semi-structurées (*ex.*, XML [W3C98]), vectorielles [BYRN99] ...

Chaque contexte requiert des besoins spécifiques en matière de manipulations, interactions et concepts. Un mauvais choix de modélisation peut avoir de lourdes conséquences sur la gestion de la base de données. Cela peut augmenter la complexité d'écriture des requêtes soumises à la base de données (*ex.*, série temporelle en relationnel), voire réduire les possibilités d'expression. De plus, ce choix impacte forcément l'optimisation de ces requêtes en empêchant l'utilisation de techniques adaptées à certains modèles, comme les indexes par exemple.

Mes travaux de recherches reposent naturellement sur ce socle. Les projets de recherche auxquels j'ai participé proposent des contextes et des données distincts :

(1) Flux RSS : Ces flux produisent des items dans un format XML en continu, dont le contenu est principalement composé de texte. La modélisation d'un flux RSS requiert une certaine compréhension de son contenu produit par les utilisateurs du Web et de la manière de les interroger dans le contexte des publications/souscriptions (Pub/Sub). Pour cela, je présenterai une analyse réalisée à la fois sur le contenu et sur le comportement des flux RSS (Section 3.1.1). Cette analyse a abouti à la définition de deux modèles de données différents auxquels j'ai pu contribuer : RoSeS (Section 3.1.2) et FiND (Section 3.1.3). Le premier hérite du modèle relationnel étendu à la gestion de flux de données, tandis que le second repose sur le modèle vectoriel pour exploiter la forte composante textuelle des items. Nous verrons en quoi chacun répond à des besoins différents dans le cadre du Pub/Sub.

(2) Les assets graphiques : Dans le cadre des jeux vidéo, la gestion des assets graphiques réalisés par les graphistes est un réel enjeu. Il faut gérer le contenu multimédia (couleurs, textures, formes, 3D, vidéo...), leurs compositions, les métadonnées (dessinateur, compagnie, jeu, catégories, taille, types...) pour permettre de définir une plateforme de manipulation et de recherche d'assets (filtre et contenu), tout en restant extensible à toute nouvelle forme de contenu ou définition de vues sur les données. Pour cela, nous nous sommes reposés sur un modèle déductif tel que Datalog, étendu à la gestion des objets (Section 3.2).

(3) Les partitions musicales : Dans le domaine musical, les partitions sont représentées sous divers formats (MusicXML, MEI, HumDrum...) dont beaucoup utilisent le standard XML. Toutefois, le schéma associé est principalement utilisé pour des besoins de traitements (maisons d'édition de partition) impliquant un mélange entre la notation musicale et son rendu visuel. D'un point de vue bases de données, cela ne facilite pas la manipulation ou la transformation de partitions pour les musicologues. À cette fin, nous avons proposé un modèle de données reposant sur les séries temporelles, ne se focalisant que sur la notation musicale pour simplifier sa manipulation sans se préoccuper du schéma de données sous-jacent (Section 3.3).

3.1 Les flux de données et RSS

Le Web 2.0 et son écosystème ont permis de développer de manière exponentielle la quantité de données disponibles sur le Web. Un des véhicules d'information est la syndication Web avec les formats RSS [RSS03] ou Atom [Ato07]. Il permet de fournir en continu aux utilisateurs abonnés à un flux des notifications de mises à jour effectuées sur leur site Web favori. Ce flux source, producteur d'items en continu, est nommé *"feed"*. Encore actuellement, tout site Web propose de publier un flux RSS, de transformer son mur *Facebook* ou son flux *Twitter* sous ce format.

Le but du projet RoSeS est de fournir un système capable d'interroger les flux RSS en continu. Mais

avant de définir un modèle de flux de données, il est nécessaire de comprendre le comportement de RSS sur le Web et la manière de l'exploiter. Pour cela, nous avons réalisé une étude approfondie [THV+14], dont je présente un extrait dans la section 3.1.1. À la suite des résultats de cette étude, je présenterai deux manières d'exploiter le contenu des flux RSS, le premier basé sur la structure semi-structurée [CATV11b] (section 3.1.2), et le second sur le contenu textuel [HKC+16] (section 3.1.3).

3.1.1 Analyse de flux RSS

L'étude approfondie du comportement des flux RSS est basée sur un jeu conséquent acquis sur une période de 8 mois en continu entre mars et octobre 2010. Nous avons pu collecter 10 794 285 items provenant de 12 611 flux. De ces données, nous nous sommes focalisés sur le taux de publication (vitesse, variation, catégorisation et caractérisation), le contenu des flux et des items (balises XML, texte, types, doublons), une analyse du vocabulaire utilisé (taille, évolution, caractérisation, rangs des termes). Nos conclusions confirment certains précédents travaux sur le Web avec quelques variations, mais nous avons également proposé de nouvelles caractérisations, voire plus précises, des flux RSS. Je ne présenterai ici que l'analyse de la structure des items et leurs tailles en nombre de mots, ainsi que l'évolution du vocabulaire, qui nous serviront dans la suite de cette thèse.

La structure des items RSS. Notre analyse empirique sur les balises XML des flux RSS est présentée dans le tableau 3.1. Elle montre qu'un grand nombre de balises prévues dans le schéma initial ne sont pas dans l?ensemble utilisées dans les items. Alors que les titres et descriptions se retrouvent dans la quasi-totalité des items, les dates de publications sont absentes dans 20% des cas, et l'identification de la langue manquante dans 30% des flux. Près de deux tiers des items ne sont pas catégorisés, de même l'auteur est présent dans moins de 8% des items. Nous pouvons également noter que 16% des flux contiennent des erreurs combinant des balises RSS et Atom. Toutefois, les flux RSS sont caractérisés par une forte prédominance de contenu textuel, disponible en grande majorité dans le titre et la description.

title	**link**	**pubDate**	**desc.**	**Language**
99.82%	99.88%	80.01%	98.09%	69.14% (feed)
author	**category**	**GUID**	**ext.**	**RSS/Atom**
7.51%	33.94%	69.50%	29.73%	16.48%

TABLE 3.1 – Taux de balises renseignées dans les items RSS

Taille des items. Des résultats précédents, nous nous sommes alors intéressés au contenu des items en mesurant la taille de ceux-ci en nombre de mots, extraits des balises : TITLE et DESCRIPTION. Cette information nous permet de mieux estimer la complexité de techniques d'indexation, de calcul de similarité ou d'analyse de données. En moyenne, un item contient 52 termes, dont 36 mots distincts.

La figure 3.1 montre la fonction de répartition cumulative de la taille des items en nombre de mots. Celle-ci a été réalisée après caractérisation de différents types de flux (en fonction de leur contenu/provenance). Nous pouvons constater la longue traîne pour chaque type, identique au comportement des pages Web [WZ05]. Il est important de noter que 51,39% des items ont une taille comprise entre 21 et 50 termes, et 14% entre 8 et 20. Nous pouvons constater trois principaux comportements : *(i)* les flux de presse contiennent des items plus longs, avec 80% des items entre 20 et 60 termes, et seulement 10% des items de moins de 20 termes ; *(ii)* les blogs avec des items très longs avec 30% des items entre 50 et 80 mots, et 20% de plus de 80 termes ; *(iii)* les autres types de flux ont 30% des items avec moins de 20 termes, et 75% avec moins de 50 termes. Ce qui montre que les items sont rarement grands, sauf dans le cas de sites Web dédiés aux informations. Par ailleurs, nous pourrons constater deux paliers pour les flux commerciaux à 6 et 8 termes. Ils sont principalement produits par des items avec un motif fixe (seulement un ou deux termes changent).

Taille du vocabulaire. L'évolution de la taille du vocabulaire provenant du contenu des items est illustrée dans la figure 3.2. Nous avons pu identifier des comportements distincts pour le vocabulaire connu dans

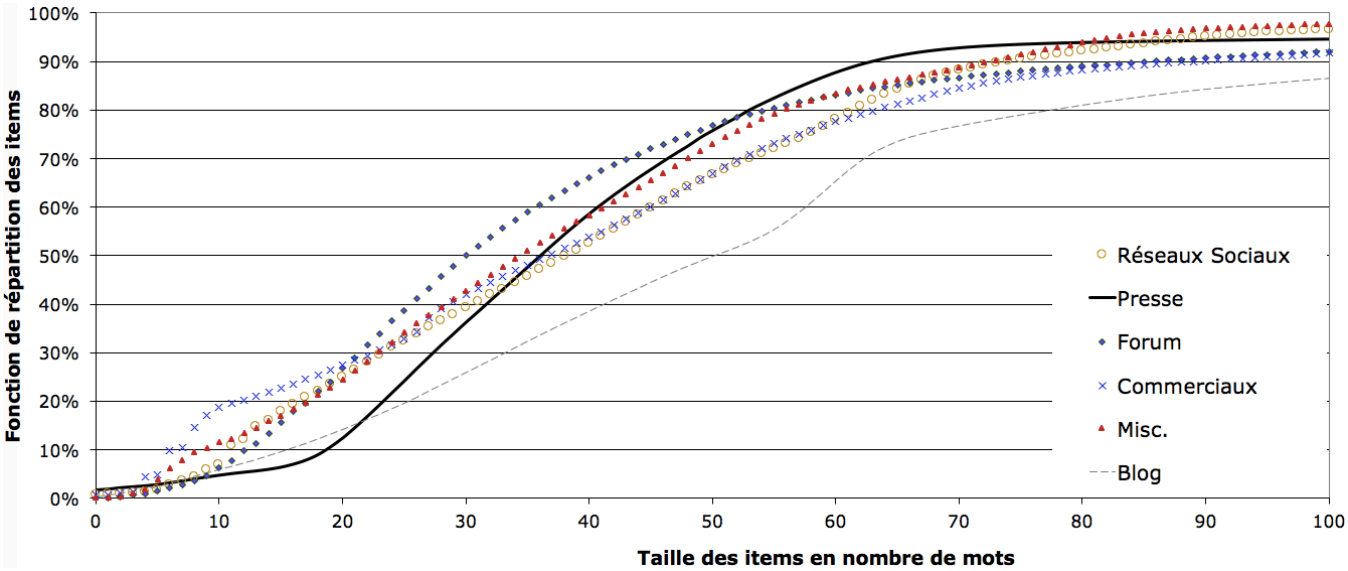


FIGURE 3.1 – Fonction de répartition cumulative de la taille des items en nombre de mots

WordNet [Mil95] (dénommé par $V_{\mathcal{W}}$) avec 61 845 mots, et pour le vocabulaire restant ($V_{\overline{\mathcal{W}}}$) avec 1 475 885 termes (acronymes, entités nommées, fautes, etc.). Naturellement, la taille de $V_{\overline{\mathcal{W}}}$ évolue beaucoup plus rapidement que $V_{\mathcal{W}}$.

Traditionnellement, l'évolution de la taille du vocabulaire est caractérisée par une loi de *Heap* [BYRN99, MRS08], définie par la formule suivante : $|V(n)| = K \times n^{\beta}$, avec n pour le nombre d'items, tandis que K et β sont des constantes (valeurs dans l'ensemble $[0,1]$) dépendant des caractéristiques du corpus textuel à analyser [WZ05]. β détermine la vitesse de l'évolution du vocabulaire avec une valeur entre 0,4 et 0,6 pour des corpus classiques [BYRN99]. β est largement affecté par l'évolution de $V_{\overline{\mathcal{W}}}$ car $V_{\mathcal{W}}$ a une taille bien plus petite. Avec une valeur d'exposant de 0,675, nous trouvons une valeur plus haute que dans la littérature [BYRN99, MRS08], indiquant une croissance plus rapide, due en majorité aux erreurs de langage et acronymes. Ce comportement correspond aux tendances observées sur des vocabulaires extraits de corpus générés par des utilisateurs (*ex.*, requêtes Web [ZMTL01, SMK05]). Pour finir, la constante K permet de prendre en compte la forte croissance des vocabulaires en début de processus, en particulier pour les termes les plus populaires.

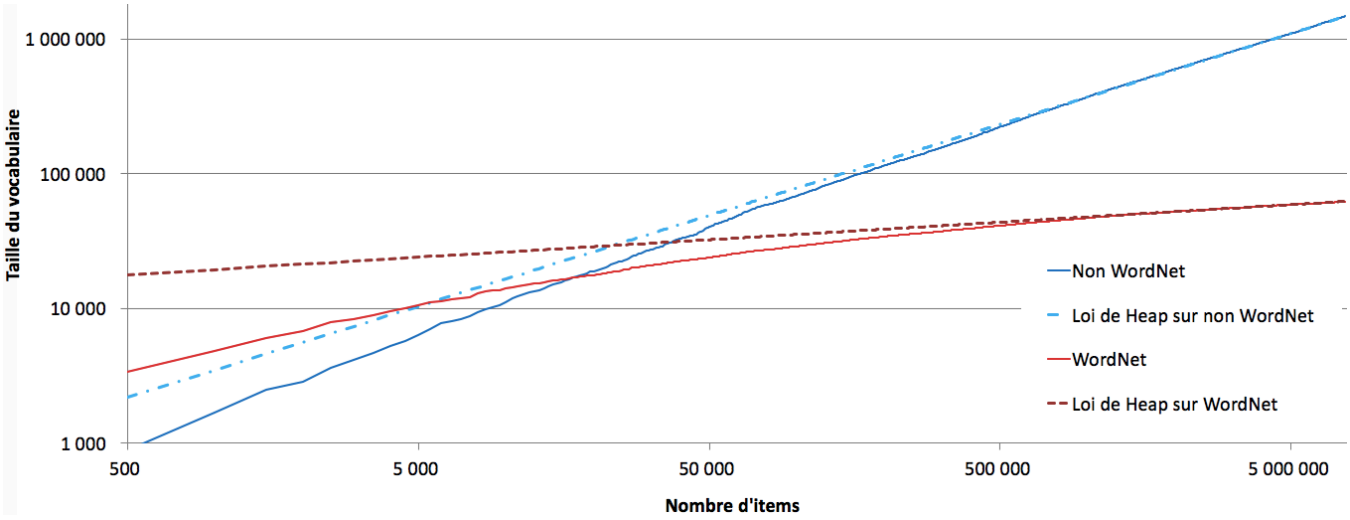


FIGURE 3.2 – Évolution de la taille des vocabulaires $V_{\mathcal{W}}$ et $V_{\overline{\mathcal{W}}}$

Conclusion. Notre analyse sur les flux RSS a permis de constater que les items sont plus grands que les annonces publicitaires (4-5 termes [KCM09]) ou les tweets (~15 termes tout au plus [Pre09]), mais plus petits que les blogs (250-300 termes [MZ07]) ou les pages Web (450-500 termes [LC06]). La taille des items dépend principalement du type de flux considéré.

Concernant l'analyse du vocabulaire, celui-ci contient une grosse proportion d'imperfections liées au Web, pouvant impacter les systèmes de filtrages par le contenu, aussi bien dans leur conception que leur performance. La croissance du vocabulaire est bornée par une loi de Heap, rendant nécessaire une sélection des termes p leurs occurrences. Par ailleurs, une étude complémentaire nous a permis de montrer que le rang des 500 premiers termes du vocabulaire n'évoluait que très rarement au cours du temps, ce qui facilite la conception d'un système basé sur le rang des termes.

3.1.2 Flux de données XML

L'étude que je vous ai présentée a permis de mieux comprendre le comportement des flux et de leurs items. Ainsi, il devient naturel de vouloir fournir un moyen d'exploiter la structure des items afin de proposer un langage d'interrogation riche dédié aux flux de données.

Le modèle de données RoSeS repose sur les modèles traditionnels des flux de données, tout en proposant des choix de modélisation adaptés aux formats de syndication du Web avec les formats RSS ou Atom et des possibilités d'agrégation. Pour ce faire, il faut tout d'abord définir les notions de temps, d'items et de flots d'items. J'ajouterai à ceux-ci deux autres types de données : les fenêtres et les annotations. Les fenêtres permettent de définir des vues instantanées sur un flux d'items qu'un opérateur "traditionnel" peut manipuler que nous étudierons dans la section 4.1.1, alors que les annotations proposent d'associer à un item des propriétés sans avoir à modifier l'item d'origine ; ce n'est donc pas une jointure à proprement parler.

Le modèle de données définit donc les types suivants : les flux sources ou "*feed*", les flux, les items, les fenêtres et les annotations.

Définition 3.1.1 (*Feed* RoSeS – f)**:** Un flux source RoSeS ou "*feed*" correspond aussi bien à l'abonnement à une source RSS/Atom externe que la publication d'un flux (virtuel).
Un *feed* est défini par le couple $f = (d, s)$, où d est sa description, et s le flux RoSeS. La description est un n-uplets dont les champs sont renseignés par les propriétés d'origine du *feed* : titre, description, URL, etc.

Définition 3.1.2 (Item RoSeS – i)**:** Un item RoSeS est composé de données provenant des items RSS/Atom. Il est composé d'un ensemble de pairs clés/valeurs provenant des balises RSS/Atom, avec une correspondance si nécessaire : *title*, *description*, *link*, *author*, *pubDate*, etc. Ce format permet d'y intégrer les extensions de schéma.

RSS étant plus largement répandu, nous reposons sur cette structure avec une correspondance des balises Atom. Comme nous l'avons vu durant l'étude, les balises restantes et les extensions de schéma sont assez peu utilisées et la principale source d'information reste le texte.

Définition 3.1.3 (Annotation RoSeS – a)**:** Une annotation RoSeS associe un item à un ensemble d'items. Les annotations sont produites par jointures entre deux flux RoSeS.
Une annotation est un ensemble de couples $a = (j, A)$, où j est l'identification de la jointure ayant produit l'annotation, et A un ensemble d'items.

La production des annotations sera développée avec l'opérateur de jointures de flux RoSeS dans la section 4.1.1.

Définition 3.1.4 (Flux RoSeS – s)**:** Un flux RoSeS est un flux de données d'items RoSeS annotés. Il est composé d'un ensemble d'éléments (éventuellement infini) $e = (t, i, a)$, où t est l'estampille de création de l'élément, i un item RoSeS, et a une annotation. L'ensemble des éléments produits à une estampille t est fini.

Définition 3.1.5 (Fenêtre RoSeS – w)**:** Une fenêtre RoSeS représente le contenu d'un flux avec l'ensemble des items valides sur une période. Une fenêtre w est un ensemble de couples (t, l), où t est une estampille, et l est l'ensemble des items du flux s valides pour t.

Il est à noter que (i) t ne peut survenir qu'une fois par fenêtre, (ii) l ne peut contenir que les items existants avant t (ou pendant). Nous pourrons utiliser la notation $l = w(t)$ pour désigner l'ensemble des items de la fenêtre à l'instant t. Les fenêtres RoSeS sont uniquement glissantes (voir figure 2.1), et peuvent être de deux types : basé sur le temps (*time-based*) ou basé sur l'activité des items (*item-based*) .

Les fenêtres sont utilisées dans RoSeS uniquement dans le cadre des opérations de jointures entre flux.

Le but de ce modèle de données est de permettre la manipulation des items, comme dans une base de données, tout en favorisant le traitement des items en temps réel sans blocage. En effet, les opérateurs vont traiter en continu les flux, en produisant de nouveaux items par filtrage, fusion, ou jointure d'annotation. Mais, le point clé reste le traitement en temps réel des items.

3.1.3 Flux de données textuels

Contrairement au modèle de données précédent, ce second modèle s'intéresse uniquement au contenu textuel des items RSS. Le modèle FiND se focalise sur les notifications liées à des mots-clés. De fait, les items sont simplifiés à des ensembles de termes, et le flux d'information à un large "*Feed*" produit par l'union de sources.

Ainsi, les items, le "*Feed*" et les souscriptions reposent sur un modèle de données composé de termes extraits des vocabulaires des items : $V_{\mathcal{W}} + V_{\overline{\mathcal{W}}}$.

Définition 3.1.6 (Item FiND – I)**:** Un item FiND est un ensemble de termes distincts : $I = (t_1, t_2, t_3, t_4)$ non triés.

Définition 3.1.7 (Le "*Feed*" FiND – $\mathcal{I}$)**:** Le flux source FiND est défini par $\mathcal{I} = [I_1, I_2, ...I_m]$, où les items I sont ordonnés par leur ordre d'arrivée temporelle : $\forall x < y \Rightarrow t(I_x) < t(I_y)$, pour lequel $t(I)$ représente l'estampille temporelle d'un item.

Définition 3.1.8 (Souscription FiND – s)**:** Une souscription FiND est un ensemble de termes distincts : $s = (t_1, t_2, t_3, t_4)$ non triés.

L'information produite par différentes sources RSS et fusionnée dans un seul "*Feed*" permet à l'utilisateur de ne pas avoir à connaître une source (URL) pour avoir l'information qu'il recherche. Il n'a qu'une seule souscription à fournir, et il sera alors notifié par tout item correspondant à sa souscription (voir section 4.1.2). Toutefois, ce choix implique des traitements optimisés pour gérer un large flux d'items de 36 termes en moyenne sur une base de millions de souscriptions.

Pondération des termes dans un flux de données textuelles. Afin d'améliorer la qualité des correspondances avec les souscriptions, il est nécessaire de rajouter des correspondances partielles en associant à chaque terme un poids. Le choix de cette pondération est prépondérant pour calculer un score de pertinence pour chaque item. Ce choix repose aussi bien sur des critères de temps de calcul que de pertinence. Dans la littérature, nous retrouvons le TF/IDF [BYRN99] avec les fréquences des termes, la valeur de discrimination d'un terme nommée TDV [SWY75], ou encore le "Term Precision" [BS74].

Le choix s'est tourné sur la fonction de pondération *TDV* qui se trouve être la plus adaptée à notre vocabulaire orienté "syndication Web". En effet, les items sont courts, ce qui implique un faible impact d'un TF, et de fait une surreprésentation de l'IDF dans le score final (démontré dans les expériences – section 5.1.3). De plus, le *TDV* mesure à quel point un terme favorise la distinction des items entre eux (*c.-à-d.*, influence du terme sur l'entropie du corpus). Ainsi, en considérant notre ensemble de souscriptions, ni les termes trop fréquents (présents dans de nombreuses souscriptions, donc peu discriminants) ni les termes peu communs (peu présents dans les souscriptions, peu de notifications) ne peuvent avoir une valeur de *TDV* élevée. Pour finir, le temps de traitement est direct puisqu'il suffit de calculer la somme des poids pour chaque souscription (section 4.1.2).

Définition 3.1.9 (TDV – $tdv(t_k)$)**:** La valeur de discrimination d'un terme t_k est définie par la différence entre la densité, produite par la matrice d'occurrences, avec le terme et sans le terme t_k. Grâce à une fonction de similarité entre deux items $sim(I_1, I_2)$ (cosinus, distance euclidienne, etc.), la densité est la

moyenne des similarités entre tous les items :

$$\Delta(\mathcal{I}) = \frac{1}{|\mathcal{I}| \times (|\mathcal{I}| - 1)} \sum_{i=1}^{|\mathcal{I}|} \sum_{j=1 \wedge j \neq i}^{|\mathcal{I}|} sim(I_i, I_j)$$

Ainsi, le TDV du terme t_k est :

$$tdv(\mathcal{I}, t_k) = \Delta(\mathcal{I} - \{t_k\}) - \Delta(\mathcal{I})$$

Pour plus de simplicité, nous la noterons $tdv(t_k)$ plutôt que $tdv(\mathcal{I}, t_k)$ lorsqu'il n'y aura pas d'ambiguïté. Maintenant que nous avons défini des poids, chaque poids de terme d'un item ou souscription est normalisé par la somme des TDV de cet item/souscription.

Mises à jour de TDV. Bien que le TDV soit adapté aux filtrages en continu, il faut toutefois produire cette valeur, et son calcul est très coûteux. Initialement, cela nous a pris deux jours pour produire les TDV à partir des 10 794 285 items. Même si cette valeur n'évolue qu'assez peu d'un mois à l'autre, la présence des termes sur des données en continu peut avoir un impact significatif sur certains mots, et les notifications associées. Afin de compléter notre modèle de données, nous avons étudié l'optimisation du calcul du TDV, de manière incrémentale et distribuée [1]. Trois principales approches existent dans littérature :

— *Naïve* [Wil85]. Le TDV d'un terme t_i est calculé tel que défini ci-dessus. Il faut produire la densité d'une collection de documents à partir de la somme de tous les pairs de similarités entre les documents, une fois avec tous les termes, et la seconde sans le terme t_i.

— *Centroïd* [Wil85]. Cette méthode simplifie le calcul en créant le *centroïd*e des documents de la collection. C'est un vecteur contenant la somme des occurrences pour chaque terme. Le TDV est alors calculé en faisant uniquement la similarité de ce *centroïd*e avec tous les documents.

— *Clustering* [CO90]. En utilisant une approche de clustering, cette méthode exploite le fait que deux documents présents dans un même cluster ont une plus grande probabilité de répondre à une même requête (mêmes mots en commun). L'algorithme de clustering détermine le nombre de clusters en créant une matrice de probabilités de corrélations entre les termes et les items. Le nombre de clusters est alors la somme des probabilités pour un terme donné. Le TDV est ensuite déterminé par la différence du nombre de clusters avec et sans le terme. En effet, le nombre de clusters est inversement proportionnel à la densité de la collection.

Toutefois, ces techniques sont dédiées à un calcul intégral sur une collection de documents statiques. Elles ne sont donc pas adaptées à un contexte évolutif. Notre extension porte donc sur une version incrémentale.

Pour cela, nous avons travaillé sur une taille de collection fixe pour garantir un calcul incrémental simplifié. Ainsi, pour chaque nouvel item, le plus ancien est alors retiré de la collection. Les changements suivants sont alors apportés aux algorithmes précédents : 1) la méthode `naïve` requiert l'annulation de l'ancien item et le calcul du nouvel item avec tous les autres, 2) la méthode `centroïde` demande une mise à jour du centroïde en retirant l'ancien et ajoutant le nouveau, puis recalculer la similarité, 3) et la méthode de `clustering` implique un recalcule du nombre de clusters en modifiant la matrice de corrélations.

La comparaison des différentes complexités des algorithmes est présentée dans le tableau 3.2. N est le nombre de termes, et M le nombre d'items. La première ligne correspond à l'algorithme de la littérature tandis que la seconde donne la version incrémentale. Même si les dimensions M et N sont grandes, M reste la plus grande avec un facteur de 100. Nous pouvons noter la forte corrélation entre les documents et le calcul de leurs similarités ($M^2N^2 \rightarrow MN^2$), même si les deux méthodes optimisées tentent de le réduire. Les gains sont obtenus par combinaisons de matrices ou production de résumé du contenu, réduisant le calcul effectué dans la seconde étape de l'algorithme ($MN^2 \rightarrow MN$).

Naturellement, ces valeurs de TDV doivent être mises à jour régulièrement. Nous avons donc cherché à savoir s'il valait mieux tout recalculer ou bien améliorer les techniques précédentes avec une mise à jour

1. Travail effectué dans le stage M2 de Ouassim Lalaoui

Méthodes	**Naïve**	**centroïde**	**Clustering**
Algorithme classique	$2M^2N^2 + 2M^2N$	$2MN^2 + 3MN$	$MN^2 + 3MN$
Algorithme incrémental	$2MN^2 + 2MN$	$2MN^2 + 2MN + N$	$2MN^2 + MN + 2N$

Table 3.2 – Complexité des différentes méthodes de mise à jour de TDV

du TDV incrémentale. La méthode Naïve ne demande que le calcul de similarité du nouvel item avec le corpus, tandis que le centroïde requiert la mise à jour du centroïde ensuite le calcul est identique. Pour la méthode clustering, un recalcule des probabilités des matrices est nécessaire pour produire le nombre de clusters. Cette amélioration n'a pas été concluante.

Toutefois, la version incrémentale peut être améliorée grâce à une distribution de l'algorithme [dMT18]. Nous avons donc implémenté chaque méthode en Map/Reduce sous `MongoDB`. En distribuant sur 16 serveurs, nous avons pu évaluer l'augmentation du temps de calcul en fonction du nombre de documents à traiter pour les TDV. Nous avons pu clairement constater que la seule technique capable de passer à l'échelle se trouvait être la méthode centroïde dont l'augmentation de temps restait linéaire dans un contexte distribué. En effet, le calcul de similarité avec le Centroïd est totalement distribuable (partie *Map*) et facilite le regroupement de tous les TDV (partie *Reduce*). Ce qui n'est pas le cas des méthodes naïves et clustering.

3.2 Modélisation d'assets graphiques

Le projet `Polymathic` a pour but de créer une plateforme de vente d'assets graphiques, sur laquelle les différents acteurs peuvent interagir simultanément. Cette plateforme doit pouvoir permettre de gérer des contenus divers à plusieurs facettes, effectuer des catégorisations, des filtrages sur contenu ou métadonnées, voir même la combinaison de tout cela. Elle doit également produire des données dérivées avec des versions altérées, des combinaisons d'assets ou des extractions de caractéristiques.

De fait, nous avons pris le parti de modéliser ces données complexes pour permettre l'intégration de ces fonctionnalités dans la base de données plutôt que de le déléguer à la plateforme. Ce qui est principalement effectué, mais partiellement dans les solutions de gestions de contenus comme Alfresco [2] ou Nuxeo [3]. Les caractéristiques de l'application que nous cherchons à mettre en avant sont :

— La `recherche par le contenu` ; images, 3D, capture de mouvements, etc. Également la recherche textuelle, mais aussi les filtrages sur les métadonnées ; format de fichiers, date de publication, taille de l'image, auteur, compagnie, catégorie, etc. Dans les solutions de gestion de contenu, seules les recherches de métadonnées et de textes sont supportées. La recherche par le contenu demande une gestion particulière à cause de l'extraction des descripteurs d'images afin de faire la correspondance entre l'asset graphique et les caractéristiques modélisées en espace vectoriel, ainsi que la définition de fonctions de similarités pour trouver les assets les plus proches de "l'image requête".

— La `gestion flexible et dynamique de collections` ; favorisant la navigation dans une hiérarchie de catégories. Les techniques utilisées reposent essentiellement sur la gestion de facettes sous *SolR* [4].

— Une `intégration transparente` pour l'utilisateur final ; que ce soit en lecture ou écriture d'assets dans son espace de travail, le but étant d'effectuer des transferts simplifiés entre le serveur et la plateforme utilisateur. Les solutions existantes effectuent des synchronisations avec une présentation des données statiques ne pouvant être changée en fonction des besoins de l'utilisateur.

Ce mémoire ne traitera pas des méthodes d'extraction de descripteurs d'images ou de la pertinence des recherches, ce travail ayant été effectué par mes collègues de l'équipe Vertigo. Le modèle de données et la conception du système sont définis de telle sorte d'être indépendant de toute gestion d'objets. Ainsi, les techniques de gestion d'images 2D, 3D ou de capture de mouvements sont intégrés comme modules à

2. `https://www.alfresco.com`
3. `https://www.nuxeo.com`
4. `https://lucene.apache.org/solr/`

la plateforme, qui peut être étendue simplement.

Ainsi, le modèle de données doit être particulièrement flexible et riche, capable de supporter un large panel d'opérations. Cela impliquera une forte complexité lors de l'évaluation des requêtes afin d'intégrer toutes les manipulations possibles. Le modèle de données que nous avons adopté est basé sur la gestion de collections de documents semi-structurés favorisant l'intégration multiple de différents types d'informations avec un schéma flexible.

Définition 3.2.1 (Types Polymathic)**:** Les types Polymathic sont des instances de la syntaxe :

$$\tau = \textbf{Coll} \mid \textbf{AType} \mid \textbf{I} \mid [\tau] \mid \{p_1 : \tau, ..., p_n : \tau\}$$

pour laquelle $p_1, ..., p_k$ sont des propriétés, $\{p_1 : \tau, ..., p_n : \tau\}$ sont le typage d'un objet,$[\tau, ..., \tau]$ sont des listes d'éléments, et **I** est un identifiant. **AType** correspond au type atomique (c.-à-d., types standards tels que chaînes, entiers, binaires, etc., ou contenus multimédias tels que **pdf**, **bib**, **jpeg**, etc.). Enfin, **Coll** est le nom d'une collection.

Les types permettent de qualifier les documents des collections, mais également les fonctions.

Définition 3.2.2 (Documents Polymathic)**:** Les documents sont des objets JSON pouvant stocker des références, des constructeurs d'imbrication d'objets ou de listes, et sont identifiés par un identifiant unique dans l'ensemble de la base de données. Une référence est une clé étrangère vers un autre document de la base, définie par sa collection et son identifiant.

Exemple 3.2.1 : Le document ci-dessous illustre les notions d'identifiants &addr$X$, d'imbrications (*authors*), de listes (*tags*), et de référence (*category*, *image*). Le contenu multimédia est stocké dans la collection 2dImage, ce qui fait que ce document est une description de cette image.

```
{
        "id": &addr1,
        "title": "Caesar's throne",
        "authors": [{"person": Person(&addrBC), "role": "graphist"}],
        "tags": ["chair", "antique"],
        "category": Category(&addr109)
        "image": 2dImage(&addr98)
}
```

Définition 3.2.3 (Collections Polymathic)**:** Une collection est un ensemble fini de documents respectant un même schéma. Le schéma de la collection est un simple objet typé, avec l'identifiant pour seule propriété obligatoire.

Exemple 3.2.2 : Ce document représente le schéma de la collection 2dAsset, illustrant le document de l'exemple 3.2.1.

```
{
        "id": I,
        "title": "string",
        "authors": [{"person": Person, "role": "string"}],
        "tags": ["string"],
        "category": Category,
        "content": 2dImage
}
```

La collection 2dImage a pour schéma {id:I, content: bits, type: jpeg}.

Définition 3.2.4 (Fonctions Polymathic)**:** Le typage d'une fonction Polymathic est défini par la forme :

$$\{p_1^* : \tau_1, \ldots, p_k^* : \tau_k, p_1^+ : \tau_m\}$$

pour laquelle $\{p_i^* \mid 1 \leq i \leq k\}$ et p_1^+ correspondent respectivement aux typages des paramètres d'entrées et de sorties de la fonction.

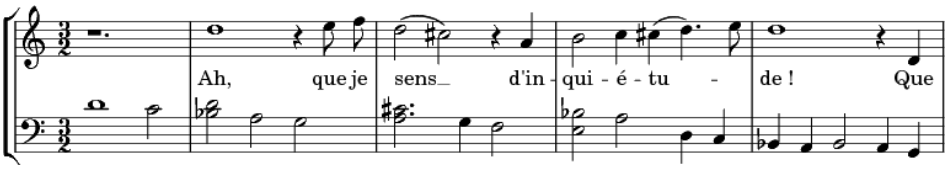


FIGURE 3.3 – Exemple de partition "*Ah que je sens d'inquiétude*" d'*Antoinette Deshoulière* (1694)

Par exemple, la fonction RGB, permettant de produire un descripteur de couleurs à partir d'une image, a pour typage : `{image*: bits, features+: [double]}`. Cette déclaration de fonctions sous forme de schéma permet de définir la signature lors de l'importation de procédures externes. Cette signature repose sur le passage de paramètres (entrée/sortie) au format JSON, pouvant être utilisée sous forme de services Web. Dans notre contexte, les fonctions d'extractions de descripteurs 2D et 3D ont été implémentées en Java et interfacées grâce à un service REST.

3.3 Séries temporelles musicales

La musique peut être considérée comme une organisation temporelle de sons. Cela peut dans la plupart du temps se résumer à des séquences d'événements musicaux, où chaque événement correspond à la production d'un son, associé à une fréquence et une durée.

Nous pouvons retrouver la musique sous forme de fichiers audio, dont la structure musicale et les voix sont difficiles à extraire de manière précise. Mais le contenu musical peut être encodé sous un langage d'échange qui a évolué durant de nombreux siècles : les partitions de musiques. De nombreuses bibliothèques musicales "DSL" (*Digital Score Librairies*) de partitions sérialisées existent et permettent de les stocker sous différentes formes d'encodage comme MusicXML [Goo01] ou MEI [Rol02, MEI17]. Ces encodages proposent une multitude de possibilités pour la notation et la représentation musicales, permettant une gestion et une manipulation de partitions illimitées. Toutefois, comme nous l'avons vu dans l'état de l'art, les techniques de manipulation sont plus orientées sur des programmations procédurales que de l'interrogation par des requêtes, voire reposent uniquement sur l'encodage des partitions (non sur les concepts musicaux), à l'instar des bases de données.

C'est sur ce constat que nous avons proposé un modèle de données pour la notation musicale [FSRT16b, FSRT16c], favorisant la manipulation de partitions musicales, sans nous préoccuper de la couche physique : l'encodage. Pour représenter le *contenu musical* dans la notation comme illustré dans la figure 3.3, nous devons respecter deux propriétés fondamentales :

(1) *Aucun chevauchement d'événements* produits par deux notes distinctes. Si l'on considère une *voix* comme concept clé pour une série temporelle d'événements musicaux, un seul événement se produit à un instant t.

(2) La *synchronisation de voix* est un aspect essentiel de la notation musicale, puisque la structure d'une partition musicale est dite *polyphonique* lorsqu'elle représente une combinaison de plusieurs voix. Et ces voix sont synchronisées temporellement, cela se représente graphiquement par un alignement vertical des notes de musique.

Le rendu visuel d'une partition prend en compte d'autres symboles tels que les clefs, les portées, etc. dont l'utilisation correspond à un problème de lecture pour l'interprète et non de la lecture musicale. De fait, ces aspects de rendus sont ignorés dans ce modèle, à l'instar des feuilles de styles pour XML.

3.3.1 Un modèle de données pour la notation musicale

Le contenu musical, tel que présenté précédemment, peut être modélisé de manière abstraite. Ainsi, une partition "*vScore*" est un ensemble de voix synchronisées, et les voix sont des séries temporelles d'événements. Cette structure favorise les manipulations à l'aide d'opérateurs de transformation.

Définition 3.3.1 (Le temps – t_i) **:** Le domaine temporel $\mathcal{T}$ est un ensemble discret, dénombrable et ordonné isomorphique à $\mathbb{Q}$.

Je repose ici sur le fait qu'il existe une unité de temps permettant de représenter le plus petit intervalle

de temps entre deux événements musicaux. Dans notre exemple, celle-ci est une huitième de note, et chaque mesure "3/2" peut être décomposée en 12 unités de temps. Une série temporelle $\mathcal{P} = \{I_1, \cdots, I_n\}$ est donc un ensemble d'intervalles temporels $I_i = [t_1^i, t_2^i[$ (ouverts à droite) tel que $\forall i, j, I_i \cap I_j = \emptyset$.

Dans le cadre des manipulations, la notion de *fusion de partitions* est nécessaire.

Définition 3.3.2 (Fusion de partitions – $\circ$)**:** Deux partitions temporelles $\mathcal{P}_1 = \{I_1, \cdots, I_n\}$ et $\mathcal{P}_2 = \{J_1, \cdots, J_m\}$ sont dites "*fusion-compatibles*" si, et seulement si, leur union est une partition temporelle, c.-à-d., $\forall i \in [1, n], j \in [1, m]$, $I_i \cap J_j = \emptyset$ ou $I_i = J_j$.

Intuitivement, deux partitions sont *fusion-compatibles* si leur union ne produit pas de nouveaux intervalles. Par exemple, $P_1 = \{[0, 2[, [4, 6[, [8, 9[\}$ et $P_2 = \{[2, 4[, [6, 7[, [8, 9[\}$ sont *fusion-compatibles*. Tous les intervalles de leur union : $P_1 \cup P_2 = \{\ [0, 2[, [2, 4[, [4, 6[, [6, 7[, [8, 9[\ \}$ proviennent soit de P_1, soit de P_2.

Définition 3.3.3 (Événement – $a_{t_1}^{t_2}$)**:** Soit **dom** un domaine de valeurs. Un *événement* $a_{t_1}^{t_2}$, $a \in \mathbf{dom}, \mathrm{t}_1, \mathrm{t}_2 \in \mathcal{T}, \mathrm{t}_1 < \mathrm{t}_2$, représente le fait que la valeur a se produit de t_1 (inclue) à t_2 (exclu). L'ensemble des événements de **dom** est dénommé par $\mathcal{E}(\mathbf{dom})$.

Les domaines de valeurs qui nous intéressent particulièrement ici sont des sons musicaux, auxquels sont associés des opérateurs :

— Les sons (**dsound**, := :, :+ :, :- :) : représentent n tons simultanés ($n \geq 1$). Cela permet de couvrir les sons simples (les notes, $n = 1$) et les sons composés (les accords, $n > 1$).

— Les syllabes, (**dsyll**, ‖).

Le son est une notion complexe qui peut être décomposée en différents composants : hauteur, intensité, timbre. En général, la notation musicale ne prend en compte que la hauteur (*pitch*) et l'octave, traduisant une fréquence. Le timbre quant à lui peut être déduit du nom de l'instrument choisi.

Ces domaines de sons sont équipés d'opérateurs dédiés. Ainsi, il est possible d'utiliser l'opérateur harmonique Ξ (deux sons joués simultanément), l'opérateur de transposition $\updownarrow$ et l'opérateur de calcul d'intervalle :- :.

Exemple 3.3.1 : Voici quelques exemples de son extrait de la figure 3.3 :

— D''^{20}_{12} est un événement de type **dsound** traduisant la première note de la première voix [5]. La valeur D'' produit un son durant son intervalle temporel de 12 (début de la seconde mesure) à 20.

— Ah^{20}_{12} est un événement de type **dsyll** sur la même période de temps.

— $< Bes, D' >^{16}_{12}$ est un événement de deux **dsound**, un accord, dans la voix du bas entre 12 et 16.

Comme spécifié dans la définition, les valeurs des événements ne sont pas restreintes aux sons. Par exemple, 2^{16}_{12} est un événement dont le domaine est **dint**, donnant le nombre de notes (2) jouées simultanément entre 12 et 16. Ce type d'événements peut être déduit de la notation et permet d'enrichir le contenu de la partition à des fins d'analyse, interprétation, etc.

Les événements ne sont pas suffisants pour définir une partition, il faut d'abord la notion de série temporelle, appelée : *voix*.

Définition 3.3.4 (Voix – v)**:** Une voix v de type **Voice(dom)** est une fonction partielle de $\mathcal{T}$ vers $\mathcal{E}(\mathbf{dom})$ tel que : $v(t) = a_{t_1}^{t_2}$ ssi $t \in [t_1, t_2[$.

La définition d'une voix en tant que fonction partielle permet de garantir le non-chevauchement des événements dans une voix. De plus, cela définit aussi le fait que pour une voix, il n'y a pas forcément un événement pour chaque instant t, dénommé $v(t) = \perp$ (absence d'événement).

La figure 3.4 montre une représentation des trois voix de l'exemple (figure 3.3) durant les trois premières mesures (estampilles de 0 à 36). Cette modélisation n'est qu'une abstraction de la notation musicale, permettant de se focaliser sur le contenu musical plutôt que sur son rendu. L'autre aspect intéressant est le fait d'introduire la notion de "voix" ayant un sens spécifique dans le domaine musical.

Définition 3.3.5 (Synchronisation de voix – $\oplus$)**:** La synchronisation de deux voix v_1 **Voice(**$\mathtt{dom}_1$**)** et v_2 in **Voice(**$\mathtt{dom}_2$**)**, est dénommée par : $v_1 \oplus v_2$, est une voix v_3 **Voice(**$\mathbf{dom}_1 \times \mathbf{dom}_2$**)** tel que : $\forall t \in \mathcal{T}$, $e_1 \in$

5. La convention Lilypond est utilisée pour représenter la hauteur de la note

$$v_{sopr}(t) = \begin{cases} r^{12}_{0}, & t \in [0,12[\\ D''^{20}_{12}, & t \in [12,20[\\ r^{21}_{20}, & t \in [20,22[\\ E''^{23}_{22}, & t \in [22,23[\\ F''^{24}_{23}, & t \in [23,24[\\ D''^{28}_{24}, & t \in [24,28[\\ C\#''^{32}_{28}, & t \in [28,32[\\ r^{34}_{32}, & t \in [32,34[\\ A'^{36}_{34}, & t \in [34,36[\end{cases} \quad v_{lyrics}(t) = \begin{cases} r^{12}_{0} & t \in [0,12[\\ \text{Ah}^{20}_{12}, & t \in [12,20[\\ r^{22}_{20}, & t \in [20,22[\\ \text{que}^{23}_{22}, & t \in [22,23[\\ \text{je}^{24}_{23}, & t \in [23,24[\\ \text{sens}^{32}_{24}, & t \in [24,32[\\ r^{34}_{32}, & t \in [32,34[\\ \text{d'in}^{36}_{34}, & t \in [34,36[\end{cases} \quad v_{bass}(t) = \begin{cases} D^{8}_{0}, & t \in [0,8[\\ C^{12}_{8}, & t \in [8,12[\\ (Bb,d)^{16}_{12}, & t \in [12,16[\\ A^{20}_{16}, & t \in [16,20[\\ G^{24}_{20}, & t \in [20,24[\\ (A,C\#)^{30}_{24}, & t \in [24,30[\\ G^{32}_{30}, & t \in [24,32[\\ F^{36}_{32}, & t \in [32,36[\end{cases}$$

FIGURE 3.4 – Séries temporelles correspondant aux trois voix de notre exemple (mesure 1 à 3)

$\mathcal{E}(\mathbf{dom}_1)$, $e_2 \in \mathcal{E}(\mathbf{dom}_2)$, la propriété suivante est valide :

$$v_3(t) = (e_1, e_2) \Leftrightarrow v_1(t) = e_1 \ and \ v_2(t) = e_2$$

Il devient alors évident que v_3 est une voix composée, telle que : $\mathcal{P}(v_3) = \mathcal{P}(v_1) \cap \mathcal{P}(v_2)$. La synchronisation combine deux voix sur le domaine temporel en produisant un croisement de domaines de valeur.

Maintenant que nous avons défini les événements, les voix et la manière de les synchroniser, nous pouvons alors définir la notion de partition, basée sur une organisation récursive de partitions et de voix.

Définition 3.3.6 (Partition – *vScore*)**:** Une partition "*vScore*" est une structure arborescente, induite par :

— une voix est une partition ;

— si $s_1, \cdots, s_n$ sont des partitions, alors $s_1 \oplus \cdots \oplus s_n$ est une partition.

Une partition S est un n-uplet dont les composants (sous-partitions/voix) sont associés à un nom.

Notre exemple illustre bien ce concept hiérarchique, avec une synchronisation de la voix v_{sopr} et v_{lyrics} pour obtenir une partition chantée, qui est elle-même synchronisée avec la voix v_{bass}. Dans le domaine musical, nous parlons de "partie" pour définir la synchronisation de voix, et de "groupe de parties" pour la synchronisation de parties. La partition peut alors être considérée comme la synchronisation de ces groupes.

Exemple 3.3.2 : Notre exemple de la figure 3.3 peut être défini de la sorte :

— le type T_v de la première portée est `[sopr: dsound, lyrics: dsyll]` ;

— le type T_b de la partition dans son ensemble est `[vocal: T_v, bass: dsound]`.

Une instance de T_b est une fonction $\mathcal{T} \rightarrow \mathcal{E}(\mathbf{dsound}) \times \mathcal{E}(\mathbf{dsyll}) \times \mathcal{E}(\mathbf{dsound})$. On peut ainsi représenter la troisième mesure par la série temporelle ci-dessous :

$$\begin{cases} (D''^{20}_{12}, \text{Ah}^{20}_{12}, < Bb, d >^{16}_{12}), & t \in [12,16[\\ (D''^{20}_{12}, \text{Ah}^{20}_{12}, A^{20}_{16}), & t \in [16,20[\\ (r^{22}_{20}, r^{22}_{20}, G^{24}_{20}), & t \in [20,22[\\ (E''^{23}_{22}, \text{que}^{23}_{22}, G^{24}_{20}), & t \in [22,23[\\ (F''^{24}_{23}, \text{je}^{24}_{23}, G^{24}_{20}), & t \in [23,24[\end{cases}$$

Nous pouvons remarquer que la paire d'événements $(D''^{20}_{12}, \text{Ah}^{20}_{12})$ apparaît deux fois, car elle est associée à deux événements distincts de la voix v_{bass}, due à une synchronisation non homorythmique.

3.3.2 Un modèle pour une œuvre musicale

Une partition à elle seule est incomplète, car elle doit être associée à la notion d'*opus* pour compléter le modèle de données. Cet *opus* contient les informations relevant de l'œuvre musicale : titre, compositeur, date de publication, etc. Ces objets devant pouvoir être interrogés dans une base de données, le concept d'opus doit être défini, reposant sur le concept relationnel.

Définition 3.3.7 (Opus)**:** Un *opus* est un n-uplet dont les valeurs peuvent être atomiques (chaînes de caractères, entiers, flottants, etc.) ou *vScores*. Une *collection* est un ensemble d'opus.

Exemple 3.3.3 : Exemple de schéma d'un Quartet :

```
Quartet (id: int, title: string, composer: string, published: date,
                music: Score [v1: dsound, v2: dsound, alto: dsound, cello: dsound])
```

Ce schéma pourrait également être le schéma d'une collection de *Quartets*. Il contient les attributs communs (titre, compositeur), et un attribut *music* de type "vScore" énumérant les quatre voix composant ce Quartet. Cette représentation suit la lignée des modèles relationnels ne respectant pas la première forme normale [AHV95].

3.4 Conclusion

Dans ce chapitre, nous avons étudié trois types de données différents ayant donné lieu à quatre modèles de données tout aussi différents. Chaque modèle est défini à partir des besoins de manipulations sur ces données.

Les flux d'items sont fortement contraints par la gestion de séquences "infinies" d'items produits au cours du temps, propre au traitement continu des données. Les deux modèles proposés diffèrent dans leur gestion de la donnée : document vs texte. Le premier s'oriente sur un langage riche capable d'exprimer des manipulations de flux, tandis que le second s'intéresse à la pertinence du filtrage sur un langage naturel. Le typage ici est assez faible, car le contenu des données reste libre, induit par le formalisme XML.

D'autre part, la gestion d'assets graphiques repose sur des collections d'objets fortement typés, tout en restant flexibles. À l'instar du modèle relationnel, ce typage a pour but de favoriser la définition d'un langage de manipulation riche exploitant la structure des objets tout en préservant ces contraintes. La flexibilité du modèle permet quant à lui de produire des versions dérivées du contenu de nos documents et ainsi d'enrichir les collections et les possibilités d'interaction offertes par le système.

Pour finir, le modèle de données de partitions musicales est très précis et détaillé, dû au domaine musical dont la complexité rend les besoins de manipulation très contraints avec des règles strictes. La modélisation sous forme de séries temporelles permet d'intégrer simplement la gestion du temps, mais n'est pas comparable à la gestion de flux comme pour RSS (la musique induit une forte corrélation entre les événements). L'introduction formelle d'opérateurs sur les voix, le typage des événements et la structure hiérarchique des partitions permettent de définir une vue abstraite sur le contenu musical d'une partition. Cette même vue favorise l'expressivité du langage en se focalisant sur l'essentiel : la musique.

Ces quatre modèles de données, bien que différents, restent essentiels au bon fonctionnement de la base de données. Ils donnent une base solide pour la définition des manipulations possibles du langage de requête que je vais détailler dans le chapitre suivant. Par la suite, nous verrons également que les choix effectués ont un énorme impact sur les possibilités d'implémentation et d'optimisation dans la base de données.

Chapitre 4
Manipulation et interrogation de données

Toute base de données se doit de fournir un langage d'interrogation, permettant d'interagir avec les données sans se préoccuper des aspects techniques liés au stockage ou à l'optimisation. Les langages de requêtes ont été introduits [Cod90] afin de définir des manipulations de données à l'aide d'expressions de langages de haut niveau.

Le langage de requêtes repose sur le modèle de données sous-jacent. Traditionnellement, il permet d'exprimer des opérations de filtrages, d'agrégations, de transformations, de fusions de données, etc. Le modèle et le langage sont fortement corrélés dans le fait de pouvoir exprimer des manipulations adaptées au modèle, mais également sur le fait d'avoir un langage accessible pour l'utilisateur final. Le langage est donc doté d'une grammaire donnant la structure de la requête et la syntaxe de manipulation des données spécifiques au modèle (objets, temps, séquences...).

Un langage de haut niveau se caractérise par le fait de pouvoir définir ce que la requête doit produire comme résultat, et non *comment* le produire. Cette étape finale d'intégration et d'implémentation des opérateurs de manipulation, ainsi que les techniques d'optimisation associées seront développées dans le chapitre 5. Je vais donc me focaliser sur la capacité des différents langages de requêtes à exprimer des manipulations adaptées à chaque modèle de données que j'ai présenté dans le chapitre précédent.

Les langages d'interrogation, tels que SQL ou XQuery sont connus pour leur abstraction des opérations du modèle de données. Ce sont des langages déclaratifs [1] dont la syntaxe permet d'exprimer des manipulations sur les données. Toutefois, bien que ces langages puissent exprimer une grande quantité de transformations, ils ne sont pas toujours adaptés dans tout contexte. En effet, un modèle de données spécifique peut engendrer des déclarations complexes, voire impossibles à exprimer (*ex.* requêtes continues). Dans le pire des cas, l'expression d'une requête pourrait même ne pas respecter les propriétés naturelles du modèle de données (*ex.* série temporelle d'événements musicaux).

Ainsi, à tout modèle de données, la réflexion doit aussi se porter sur le langage d'interrogation qui doit être capable d'exprimer l'essence même du modèle de données tout en respectant ses propriétés. Il doit, autant que possible, être défini de manière abstraite les manipulations de données pour à la fois faciliter l'expressivité et permettre à l'optimiseur de définir le mode opératoire.

Je vais donc présenter trois familles de langages de requêtes dont la complexité repose sur sa grammaire (langage complexe), ses opérations (mixe entre le quoi et le comment produire les données) ou son implémentation (langage simple, mais dur à optimiser). Ces trois familles sont des langages :

(1) **Déductifs** : *Datalog*. Le langage de requêtes exprime des règles logiques afin de contraindre ou de générer des données. Bien que simple à exprimer, sa complexité réside dans l'implémentation et l'optimisation de ces règles.

(2) **Déclaratifs** : *XQuery* (`RoSeS` et `ScoreQL` en sont dérivés). La manipulation des données est exprimée avec un langage de haut niveau. Même si l'implémentation des opérateurs est cachée, le langage implique une notion de séquences d'opérateurs pour produire le résultat final. Cette séquence s'approche de la notion de plan d'exécution.

(3) **Naturels** : *Mots-clés*. Le langage et le modèle de données sont similaires, ils reposent sur un ensemble de mots caractérisant les données. L'expressivité est limitée, mais extrêmement simple à exprimer pour l'utilisateur final. Il repose sur un modèle mathématique, dont le principal problème est de gérer la pertinence du résultat et le moyen de l'optimiser.

4.1 Les requêtes Pub/Sub

Le paradigme Pub/Sub, contrairement aux bases de données traditionnelles, repose sur le fait que les données sont temporaires (en continu) et que les requêtes sont persistantes (stockées). Ainsi, pour être notifié, l'utilisateur définit sa requête, appelée `souscription`. Cette notification est déclenchée dès

1. À l'exception de l'extension XQuery scripting : `https://www.w3.org/TR/xquery-sx-10/`

qu'une donnée, appelée `item`, correspond au critère recherché. Ce principe impose une évaluation à la volée des items sur les souscriptions stockées.

Pour cela, je vais présenter deux approches correspondant aux deux modèles de données présentés précédemment (sections 3.1.2 et 3.1.3). La première repose sur une approche algébrique pour permettre la gestion et la composition de flux de données, avec un langage de requêtes dédié. La seconde approche s'intéresse aux méthodes de calcul de la pertinence pour le filtrage intelligent des notifications.

4.1.1 Approche algébrique pour les flux de données

Le langage de manipulation de flux RoSeS repose sur la faculté de définir des expressions algébriques (facilitant l'optimisation de requêtes), et l'intégration transparente de flux de données.

L'algèbre RoSeS. L'algèbre RoSeS défini des *publications* à partir de sources externes, mais également sur d'autres publications (composition). Cette algèbre repose sur cinq opérateurs, permettant de définir des expressions algébriques. La génération d'expressions algébriques produit de nouveaux flux de données, appelés *flux RoSeS*. Nous pouvons distinguer deux types d'opérateurs, (1) les opérateurs *préservants* (`filtrage`, `union`, `fenêtrage` et `jointure`), qui ne changent pas le contenu des items ni ne produisent de nouveaux items, et (2) les opérateurs *altérants* (`transformation`) qui produisent de nouveaux items.

Un des choix les plus intéressants du langage RoSeS est d'imposer *uniquement des opérateurs **préservant** dans le langage de **publication***. De fait, les opérateurs *préservants*, qui manipulent les items *(t,i,a)*, ont toutes les propriétés indispensables à l'optimisation de requêtes (commutativité, associativité, distributivité, etc.), et facilitent l'expressivité des requêtes (expressions algébriques normalisées). L'opérateur de jointure est lui-même très intéressant puisqu'il n'est pas bloquant pour les flux de données.

Définition 4.1.1 (Filtrage – σ)**:** Le *Filtrage* produit un flux dont les items satisfont le prédicat donné : $\sigma_P(s) = \{(t,i,a) \in s | P(i)\}$. Le *prédicat* est une expression booléenne (conjonctions, disjonctions, négations) composée de conditions sur les attributs de l'item i. Les types de conditions peuvent être simples, des fonctions sur le temps, des similarités textuelles, voire exploiter les Urls.

Définition 4.1.2 (Union – $\bigcup$)**:** L'*Union* produit tous les items des différents flux fournis en entrée de l'opérateur : $\bigcup(s_1,...,s_n) = s_1 \cup ... \cup s_n$. L'union peut être explicite (liste des flux) ou implicite à l'aide de l'opération *"collection()"* associée à un critère pour récupérer l'union des flux correspondants.

Définition 4.1.3 (Fenêtrage – ω)**:** Le *Fenêtrage* produit une fenêtre sur le flux d'entrée à l'aide du type de fenêtre associé ; $\omega_{t,spec}(s)$ et $\omega_{c,spec}(s)$ définissent respectivement des fenêtres glissantes basées sur le temps et sur les items, pour lesquelles *spec* précisent la durée (respectivement, le nombre d'items).

Définition 4.1.4 (Jointure – $\bowtie$)**:** La *jointure* prend un flux principal et une fenêtre sur un flux secondaire. Cette variante *préservante* de l'opérateur de jointure est appelée, `jointure d'annotation`, qui agit comme une semi-jointure (flux principal filtré par le contenu de la fenêtre), tout en gardant une trace des items joints sous forme d'annotations. Une jointure $\bowtie_P(s,w)$, identifiée par j, produit les items de s pour lesquels le prédicat P est satisfait par un ensemble non vide d'items I de la fenêtre w, et ajoute ceux-ci comme annotation (j,I). Plus précisément :

$$\bowtie_P(s,w) = \{(t,i,a') | (t,i,a) \in s, I = \{i' \in w(t) | P(i,i')\}, |I| > 0, a' = a \cup \{(j,I)\}\}$$

Définition 4.1.5 (Transformation – τ)**:** La *transformation* modifie chaque élément en entrée grâce à une fonction de transformation donnée : $\tau_T(s) = \{T(t,i,a) \mid (t,i,a) \in s\}$. C'est le seul opérateur *altérant*, uniquement utilisé lors de la production de souscriptions ou de sources (pas des publications).

Exemple 4.1.1 (Expressions algébriques RoSeS)**:** Les expressions algébriques suivantes illustrent les opérateurs *préservant* ci-dessus. Nous pouvons remarquer l'utilisation d'une composition entre les publications *RockConcertStream* dans la définition du flux *MusePhotoStream*.

$$\text{RockConcertStream} = \sigma_{'concert' \in desc}\ (\text{FFacebook} \cup (\sigma_{'rock' \in title}\ \text{EAnnounces}) \cup \text{FTwitter})$$
$$\text{MusePhotoStream} = (\sigma_{'Muse' \in desc}\ (\text{RockConcertStream}) \cup \text{MNews}) \bowtie_{title \sim w.title} \omega_{last\ 3\,m}\ (\text{MusePhotos} \cup (\sigma_{cat = 'rock'}\ \text{FPhotos}))$$

L'opérateur altérant de transformation est utilisé lors de la matérialisation de sources de données avec le langage d'`enregistrement` et de `souscription`. Il est à noter que cette transformation peut utiliser le résultat des jointures, et de fait enrichir le contenu des items et améliorer l'expressivité des requêtes.

Le langage de requêtes RoSeS. Il est évident qu'un utilisateur ne définit pas directement des expressions algébriques. Un langage de requêtes est nécessaire pour définir les instructions à appliquer sur les flux. Elles sont décomposées en trois parties : l'enregistrement de flux sources "`register feed`", la publication de flux/création "`create feed`" et la souscription aux notifications d'un flux "`subscribe to`". Pour des raisons de simplifications, je résumerai ici les composants du langage et l'illustrerai à l'aide d'exemples.

Le langage de `publication` est à la fois expressif et simple d'utilisation, intégrant simplement chaque opération. Il est également approprié à l'optimisation multi requêtes comme nous pourrons le constater dans le chapitre suivant. Une requête de publication doit contenir trois clauses : 1) la clause **from** obligatoire qui spécifie le flux d'entrée, appelé flux principal, 2) des clauses **join** optionnelles, précisant chacune une jointure sur un flux secondaire produisant des annotations sur les items joints, 3) une clause **where** optionnelle contenant les prédicats de filtrage sur le flux principal et les secondaires.

Le langage d'`enregistrement` de flux permet de définir de nouvelles sources pouvant provenir de flux externes (RSS/Atom) ou des publications internes précédemment matérialisées. Il est à noter que le langage RoSeS ne permet pas d'utiliser des transformations sur une publication sans avoir été matérialisée précédemment (`subscribe to`). Une transformation est exprimée à l'aide d'une feuille XSLT extrayant les propriétés des items RoSeS et de leurs annotations.

Le langage de `souscription` de flux permet donc de souscrire à une publication (ou source) en précisant : 1) le flux, 2) le mode de diffusion (RSS, mail, etc.), 3) la périodicité des mises à jour, et 4) éventuellement une *transformation*.

Exemple 4.1.2 (Requêtes RoSeS)**:** La requête de publication (`create feed`) suivante définit un flux de concerts de rock, dont les sources proviennent de messages d'amis (*FFacebook* et *FTwitter*), et d'annonces de concerts du flux *EAnnounces*. Cet exemple correspond à l'expression algébrique de l'exemple 4.1.1 :

```
register feed https://www.facebook.com/feeds/friends_notes.php?id=x&key=y&format=rss20 as FFacebook;
register feed https://twitter.com/statuses/user_timeline/174451720.rss as FTwitter;
register feed https://www.infoconcert.com/rss/news.xml as EAnnounces;
create feed RockConcertStream from (FFacebook | EAnnounces as $ca | FTwitter) as $r
                where $ca[title contains 'rock'] and $r[description contains 'concert'];
```

Nous pouvons composer cette publication en créant une nouvelle publication *MusePhotoStream*, dont les items parlent de *Muse*, et sont annotés avec des photos. Ces items proviennent des flux *RockConcertStream* et *MuseNews*, tandis que les photos des flux *FriendsPhotos* et *MusicPhotos* (uniquement pour la catégorie 'rock'). Cette annotation est effectuée à l'aide d'une jointure, avec une fenêtre de trois mois ayant des titres similaires (entre le flux principal et la fenêtre).

```
register feed https://muse.mu/rss/news.rss as MuseNews;
create feed MusePhotoStream from (RockConcertStream as $r | MuseNews) as $main
        join last 3 months on (MusicPhotos as $m | FriendsPhotos) with $main[title similar window.title]
        where $r[description contains 'Muse'] and $m[category = 'rock'];
```

Une matérialisation est illustrée dans ce dernier exemple sur le flux *MusePhotoStream* en appliquant la feuille de style "`IncludePhotos.xsl`". Deux souscriptions sont également proposées sur ces nouvelles publications : la première extrait les titres "RockTitle.xsl" avec un envoi par email toutes les trois heures, et la seconde produit un nouveau flux RSS rafraîchi toutes les dix minutes.

```
register feed MusePhotoStream apply 'IncludePhotos.xsl' as MuseWithPhotos;
subscribe to RockConcertStream apply 'RockTitle.xsl' output mail 'me@mail.org' every 3 hours;
subscribe to MusePhotoStream output file 'MuseConcertStream.rss' every 10 minutes;
```

4.1.2 Approche filtrage par pertinence

Le traitement Pub/Sub de flux orientés textes repose sur la notification d'items correspondant aux souscriptions stockées. Bien que simple sur le principe, le fait que les souscriptions soient temporaires ne facilite pas le problème d'inclusion pour un item "requête", dont nous étudierons le traitement en section 5.1.2. La souscription que nous avons vu dans la définition 3.1.8 est simplement composée de mots-clés qui reflètent les intérêts de l'utilisateur ; à l'instar des requêtes sur un moteur de recherche.

Dans cette thèse, je noterai $\mathcal{S}$ pour définir l'ensemble des souscriptions stockées et $|\mathcal{S}|$ le nombre de celles-ci. Chaque souscription $s \in \mathcal{S}$ est composée de termes distincts extraits du vocabulaire $\mathcal{V}_S = \{t_1, t_2, ..., t_n\}$. La taille de s est définie par $|s|$.

Comme vu dans la définition 3.1.6, un item I est également défini par un ensemble de termes. Naturellement, une correspondance item/souscription se produit si, et seulement si, tous les termes d'une souscription s sont présents dans l'item I. C'est ce que nous appelons : requête large (*broad match*).

Définition 4.1.6 (Requête large)**:** Une requête large est produite entre une souscription s et un item I si, et seulement si, $\forall t_i \in s, t_i \in I$.

Exemple 4.1.3 (Correspondances de requêtes larges)**:** Soit l'item $I = \{t_2, t_3, t_4, t_7\}$, et les quatre souscriptions ci-dessous. Seule s_4 vérifie la sémantique de requête large pour I, car celui-ci contient les termes t_2 et t_4. Pour toutes les autres, il manque au moins un terme.

$\mathbf{S_1} : t_1 \wedge t_2 \wedge t_4$, $\mathbf{S_2} : t_1 \wedge t_3$, $\mathbf{S_3} : t_2 \wedge t_5 \wedge t_7$, $\mathbf{S_4} : t_2 \wedge t_4$, $\mathbf{S_5} : t_2 \wedge t_3 \wedge t_4 \wedge t_7 \wedge t_8$

Cette définition simple facilite le traitement d'un large flux d'items sur un grand nombre de souscriptions comme nous le verrons par la suite. Toutefois, lors de nos travaux de recherche, nous avons constaté que toutes les souscriptions ne se comportaient pas de la même manière. Certaines ne généraient aucune notification, tandis que d'autres étaient totalement submergées de notifications. Il était donc nécessaire de modifier le principe de *correspondance* des souscriptions avec : des correspondances *partielles* et le filtrage par *pertinence*. Ces deux aspects reposent sur la pondération des termes avec le TDV (définition 3.1.9).

Correspondances partielles. Les requêtes larges pouvant être trop restrictives pour certaines souscriptions (parfois trop longue comme s_5, ou termes trop discriminant), il est nécessaire de relâcher cette sémantique avec une correspondance partielle. Une souscription pourra correspondre si une partie conséquente de ses termes sont présents dans l'item. Cette partie conséquente repose sur la pondération des termes comme les TDV.

Définition 4.1.7 (Correspondance partielle)**:** Une souscription correspond partiellement à un item I lorsque le score de correspondance $\mu(s, I)$ est supérieur au seuil κ. Le score de correspondance est défini à l'aide de la somme des poids des termes de s présents dans I : $\mu(s, I) = \sum_{t \in s \cap I} \rho(t, s)$, où $\rho(t, s)$ est le TDV normalisé du terme t dans s.

Expérimentalement, nous avons montré qu'un bon seuil pour κ était de 0,75 ; meilleur compromis entre taux de notifications et performances.

Correspondances pertinentes. Contrairement au problème précédent, certaines souscriptions génèrent un grand nombre de notifications et les utilisateurs se retrouvent submergés. Afin de réduire le débit de ce flux, deux approches permettent de gérer les pertinences des items notifiés en supprimant les informations redondantes, appelées filtrage par **nouveauté** [CKC+08], ainsi que la diversification des informations notifiées, ou **diversité** [DP09] (souvent réduit à un problème top k).

Le système `FiND` prend en compte la *nouveauté* et la *diversité*. Pour ce faire, nous devons garder pour chaque souscription les items déjà notifiés que nous appelons *historique de souscription* H. Chacun est un ensemble d'items, correspondant aux notifications de la souscription, ordonnées par ordre d'arrivée.

Pour commencer, la **nouveauté** a pour but d'éliminer tout item qui ne contient pas de nouvelle information au vu de l'historique de souscription. En d'autres termes, l'item I n'ait ni tronqué ni similaire à un item I' de H. Du fait que notre historique est trié temporellement, la mesure de nouveauté *new(I, I')* doit être asymétrique [ZCM02] afin de vérifier la nouveauté de I sur l'historique, et non le contraire.

Ainsi, la nouveauté d'un item I par rapport à un historique peut être vérifiée par une comparaison de I avec tous les items de H.

Définition 4.1.8 (Nouveauté item-historique)**:** Soit un historique d'items H et un item I, I est considéré comme nouveau envers H, ssi :

$$\forall I' \in H, new(I, I') \geq \alpha$$

Chacune des comparaisons doit dépasser un seuil de nouveauté α. Nous avons trouvé qu'un seuil de 50% donnait le meilleur score de qualité de résultat (Précision/Rappel).

Comme la mesure "*new*" doit être asymétrique, une distance classique comme *Jaccard* n'est pas appropriée. De fait, nous avons proposé une mesure inspirée de [GDH04], pour comparer la nouveauté d'un item avec un autre. Cette mesure calcule la couverture pondérée des termes de I sans prendre en compte les termes de I' ; c'est la somme des poids des termes présents dans I.

Définition 4.1.9 (Nouveauté item-item)**:** Soit le seuil de nouveauté $\alpha \in [0,1]$, et I, I' deux items. I est considérée comme nouveau par rapport à I' ssi :

$$new(I, I') = \frac{\sum_{t \in (I \setminus I \cap I')} tdv(t)}{\sum_{t \in I} tdv(t)} \geq \alpha$$

D'autre part, la **diversité** est un filtrage complémentaire à la redondance, car il vérifie si l'information de l'item est globalement présente dans l'historique de la souscription. Le degré de diversité d'un item est mesuré en fonction de l'accroissement de la moyenne des distances $D(H)$ entre les items "*dist (I, I')*" [DP09]. Pour diversifier l'historique, I doit être en moyenne plus distant de tous les items de H, qu'au moins un des items de H. Toutefois, pour que les mesures $D(H)$ et $D(H')$ (avec I) soient comparables, le nombre d'items doit être identique. De fait, nous avons choisi de vérifier la diversité avec I_o, l'item le plus ancien de H. Ce choix est justifié par le fait que I_o a le plus de chance d'être le plus distant de tous les items, car il contient des informations plus anciennes.

Définition 4.1.10 (Diversité item-historique)**:** Soit l'historique d'items H pour lequel $D(H)$ est la moyenne des distances entre chaque item qui le compose. Un item I augmente la diversité de H ssi :

$$D(H \cup \{I\} - \{I_o\}) > D(H)$$

avec I_o le plus vieil item de H, et :

$$D(H) = \frac{1}{|H| * (|H| - 1)} \sum_{I \in H} \sum_{(I' \in H \wedge I' \neq I)} dist(I, I')$$

Nous pouvons noter que le calcul de la diversité repose sur le calcul de la distance entre les items. Plusieurs mesures de distance pour la diversité ont été proposées dans la littérature. Nous pouvons retrouver le *Cosinus* [ZCM02], la distance *euclidienne* [DP12a, PvA08] et la distance *Jaccard* [DP12b]. Toutefois, d'autres mesures dérivées comme *Pearson*, *Dice* ou *Levenstein* ont été proposées. Au vu de notre contexte avec des items de petite taille (36 termes), la distance *euclidienne* est connue pour produire des résultats plus pertinents [BBB+10]. Ainsi, après expérimentation, nous avons pu montrer qu'une distance *euclidienne* pondérée par des TDV donnait des résultats plus pertinents pour la mesure de la diversité.

Exemple 4.1.4 (Nouveauté et Diversité)**:** Soit la liste d'items suivants correspondant à la souscription S_4 :

Historique de souscription S_4	**Items à filtrer**
$I_1 : t_2 \wedge t_4 \wedge t_7$	$I_4 : t_2 \wedge t_4 \wedge t_8$
$I_2 : t_2 \wedge t_4 \wedge t_8 \wedge t_9$	$I_5 : t_2 \wedge t_4 \wedge t_8 \wedge t_7$
$I_3 : t_2 \wedge t_4 \wedge t_3 \wedge t_9$	$I_6 : t_2 \wedge t_4 \wedge t_3 \wedge t_9 \wedge t_1 0$

Les mesures de nouveauté et diversité ignorent naturellement les termes de souscription S_4 forcément présents dans les items (t_2 et t_4). Ainsi, l'item I_4 n'est pas nouveau, car contenu dans I_2, I_5 est nouveau mais ne diversifie pas l'historique, car il est couvert par l'union $I_1 \cup I_2$, mais I_6 est à la fois nouveau et divers. À noter qu'un item peut diversifier l'historique et ne pas être nouveau si la somme des distances est plus grande que celui de l'historique tout en ayant un item similaire.

4.2 Une approche basée sur les règles

Le modèle de données de Polymathic requiert un langage capable de manipuler des objets avec un fort typage, tout en restant flexible dans la définition de nouvelles collections. Le langage doit également être capable d'effectuer des recherches sur les métadonnées, mais aussi sur le contenu multimédia, et effectuer des transformations sur les données (facettes). Le langage à base de règles Datalog [CGT89] permet de définir des règles *actives*, produisant de nouveaux documents matérialisés, ou *déductives*, pour des opérations de recherche. Je vous montrerai dans cette section comment ce langage de règles peut servir de ciment pour la cohérence des collections, nécessaires au bon fonctionnement de notre système.

4.2.1 Datalog

Les collections peuvent être stockées (*extensionnelles*) ou définies par des règles (*intensionnelles*). Cette approche est intéressante, car les collections ou les fonctions sont vues comme des prédicats P, et interprétées de la même manière :

Définition 4.2.1 (Prédicat – P)**:** Si C est une collection et d un document, le prédicat $C(d)$ est vrai, ssi d appartient à la collection.

Si F représente la fonction f, avec i et o l'ensemble des paramètres d'entrées et de sorties, alors le prédicat $F(i,o)$ est vrai, ssi $o = f(i)$.

Nous pouvons alors définir un système simple, géré entièrement à l'aide de règles *statiques* (collections) et *dynamiques* (fonctions). Nous utilisons la syntaxe standard définie pour les règles Datalog [AHV95].

Définition 4.2.2 (Règle Datalog)**:** Une règle Datalog est définie par un ensemble de prédicats $P_i, 1 \leq i \leq n$ (collection ou fonction), et une collection intensionnelle C :

$$C(v_0) :- P_1(v_1), \ldots, P_n(v_n)$$

où chaque v_i est soit un document, soit une variable. $C(v_0)$ est vrai si chaque prédicat $P_i(v_i)$ est vrai.

La syntaxe Datalog est étendue pour prendre en compte les objets complexes [NT89, AG91]. De plus, certains prédicats arithmétiques sont pris en compte ($=, \leq, \geq, !=$) pour définir des règles plus naturelles, mais également la notion d'existence dans une liste S avec le prédicat `x in S`.

Exemple 4.2.1 (Règle)**:** La règle ci-dessous produit une collection "Chair" contenant des assets graphiques de la collection Asset2d dont un des tags est un fauteuil.

```
Chair($a) :- Asset2d ($a), $tag in $a.tags, $tag = "Chair"
```

Le langage ne spécifie pas si la collection intensionnelle doit être matérialisée ou évaluée sur demande, toutefois, le système Polymathic matérialise par défaut toutes collections définies par une règle *active*. De fait, toute mise à jour d'un document contenu dans une collection intensionnelle ou extensionnelle déclenche la ou les règles associées. Les règles *dynamiques* ne sont pas matérialisées, comme la collection answer que nous verrons par la suite, mais aussi les deux collections particulières File et Folder qui sont utilisées dans le fonctionnement de notre système de fichier virtuel, détaillé dans la section 4.2.3.

Règles saines. Les collections sont des ensembles finis (naturel pour les collections extensionnelles). Mais pour le cas des collections intensionnelles, pour que les règles soient saines et garantissent une stabilisation du contenu de ces collections, il est nécessaire que ces ensembles restent finis. Malheureusement, l'utilisation de fonctions comme prédicats peut aboutir à la génération de règles non saines [AH88, SV89].

La première possibilité de problèmes vient des paramètres d'entrée d'une fonction pouvant ne pas être définis avant l'évaluation de la fonction. La seconde possibilité d'erreur est le cas des cycles, pouvant générer un nombre infini de valeurs. L'exemple ci-dessous illustre ces deux cas de problèmes.

Exemple 4.2.2 (Règles non saines)**:** Le premier cas illustre deux fonctions $F1$ et $F2$ dont le premier paramètre est l'entrée, et le second la sortie. $y n'étant pas affecté à l'origine, cette règle peut dans certains cas générer une infinité de résultats.

```
answer($x, $z) :- F1($x, $y), F2($y, $z)
```

Le second cas illustre une règle récursive qui va générer une infinité de valeurs dans la collection C.

C($b) :– **C**($a), **Add**($a, 1, $b)

Pour le premier problème, une solution est de vérifier que la variable d'entrée d'une fonction est associée à une collection comme prédicat (intensionnelle ou extensionnelle). L'ensemble de valeurs est alors fini. La stratification suivante permet de vérifier l'affectation saine des variables dans l'ensemble B :

(1) Créer une première strate S_1 avec tous les prédicats de type "collection" dans le corps de la règle ; ajouter à B toutes les variables utilisées dans ces prédicats.

(2) Créer une seconde strate S_2 avec les prédicats de type "fonction" utilisant des variables définies dans B ; ajouter à B toutes les variables en paramètre de sortie de ces fonctions.

(3) Répéter la seconde étape avec une nouvelle strate S_i jusqu'à qu'un point fixe soit atteint ($S_i = \emptyset$).

Si tous les paramètres d'entrées des fonctions sont inclus dans une des strates, alors la règle est considérée comme saine, dans le cas contraire, elle est rejetée.

La question des cycles est traitée dans le moteur d'exécution que je présente en section 5.2.1, pour éviter la génération infinie de documents : pour chaque événement déclenchant une règle, le nombre de déclenchement d'une règle est limité (trois fois en l'occurrence), si c'est le cas l'exécution s'arrête et une exception est générée. Cette méthode est standard pour les bases de données actives [WC96].

Les règles servent d'outils de haut niveau pour le système `Polymathic`. Il permet de lier les différents composants de l'application, de centraliser l'implémentation, et de garder une approche déclarative dans la gestion des documents de la base documentaire. Les règles peuvent être vues comme un *workflow* autogéré, ce qui simplifie grandement l'administration du système, ainsi que l'implémentation des tâches.

4.2.2 Recherche par le contenu

Afin de permettre les recherches par le contenu, le système `Polymathic` repose sur le concept de dérivation de contenu. Cette tâche est tout particulièrement utilisée lors de l'insertion de données dans les collections, afin de produire des miniatures pour les images, des vecteurs de description, des tags, etc. agissant comme un trigger sur ces mises à jour. L'implémentation des triggers sera explicitée dans la partie implémentation (section 5.2.1), mais le concept de dérivation permet de comprendre le workflow utilisé pour la recherche par le contenu. En effet, la dérivation de contenu étant effectuée *via* une règle, et que les données intentionnelles sont stockées, il est alors possible d'effectuer des recherches sur ces dérivations dans un prédicat de requête dynamique.

Exemple 4.2.3 (Dérivation d'une image et recherche)**:** Les descripteurs d'images sont souvent utilisés pour représenter le contenu d'une image. Ces descripteurs sont associés dans l'espace vectoriel à une fonction de distance pour pouvoir trier la pertinence des recherches. L'extraction du descripteur se fait via une fonction, comme par exemple "`RGB(content*, feature+)`" pour extraire la couleur d'une image. Une première règle matérialise ainsi l'ensemble des descripteurs de couleurs de la collection `Asset2d` :

RGBDescriptor($a.id, $fv) :– **Asset2d**($a), **RGB**($a.image.content, $fv)

La seconde règle est une requête (`answer`) de recherche par le contenu trouvant les images proches d'une image requête "$target" dont le descripteur est extrait à la volée "$fv2", puis comparé à tous Asset2d contenant le mot "Chair" et dont la distance (L2) entre les deux descripteurs est inférieure à 0,3.

answer($a, $dist) :– **Asset2d** ($a), 'chair' **in** $a.tags, **RGBDescriptor** ($a.id, $fv1), **RGB**($target, $fv2),
L2($fv1, $fv2, $dist), $dist < 0.3

Cette approche facilite la combinaison des métadonnées et des recherches par le contenu, mais également l'intégration de nouveaux types de données multimédias ou de descripteurs. Il suffit pour cela de définir la fonction d'extraction, de distance et la règle associée.

4.2.3 Requêtes dans un système de fichier

Les règles que je présente dans cette section ont pour but de faciliter l'intégration de notre système dans l'environnement de l'utilisateur. En manipulant simplement un fichier dans son espace de travail, le

système Polymathic va synchroniser automatiquement le fichier dans une base de données et appliquer la série de dérivations sur ce nouveau document. De même, tout nouveau document produit par un utilisateur sera alors synchronisé dans les espaces de travail partagés par d'autres utilisateurs.

Polymathic définie un système de fichier virtuel (VFS) à l'aide de règles spécifiques sur les documents de la base de données. Cette approche a déjà été proposée (voir [BMPT09, AMM08]), mais les possibilités d'interrogation étaient restreintes. Deux collections intentionnelles définissent le fonctionnement du VFS, avec les collections "Folder" et "File", typées de la manière suivante :

Folder ("id": I, "vfs": "string", "name": "string", "parent": **Folder**)
File ("id": I, "name": "string", "extension": "string", "content": "bitstring", "folder": **Folder**)

Ces deux collections représentent les notions de répertoires et de fichiers qui seront instanciés à la volée lors de l'interaction de l'utilisateur avec son système de fichier. Chaque répertoire ou fichier est un artefact, associé à un document de la base de données. L'implémentation de cette interaction avec le VFS et la visualisation du contenu d'un répertoire sera étudiée avec FUSE dans la section 5.2.2. Plusieurs types de VFS peuvent être définis, chacun spécifie un ensemble de règles donnant la construction de la hiérarchie de répertoires et la répartition des fichiers dans ces répertoires.

Exemple 4.2.4 (Système de fichier basé sur les catégories)**:** Pour illustrer le fonctionnement du système de fichier virtuel, nous allons prendre l'exemple d'une hiérarchie de catégories. Cette hiérarchie repose sur une taxonomie de catégories qui classe les assets stockés dans la base de données organisée sous forme arborescente avec le schéma suivant : Category{id:I, label:string, parent:Category}.

Tout d'abord, un répertoire racine (*Folder*) est instancié pour la catégorie n'ayant pas de parent, puis une règle récursive instancie la hiérarchie des répertoires.

Folder ($c.id, "categories", $c.label, null) :– **Category**($c), $c.parent = null
Folder($a.id, "categories", $c.label, $c.parent) :– **Category**($c), **Folder**("id": $c.parent.id, "vfs": "categories")

Ensuite, les assets peuplent les répertoires associés (catégorie correspondante) avec des documents *File*. Nous pouvons remarquer qu'un fichier est identifié à la fois par l'asset associé, mais aussi par le répertoire du VFS. Cela permet de définir un identifiant unique pour chaque fichier, quand bien même un même asset peut produire des artefacts dans différents répertoires (même VFS ou VFS différents). D'autre part, le nom du fichier est défini par le label de l'asset.

File([$f.id, $a.id], $a.title, $a.image.type, $a.image.content, $f) :– **Folder** ($f), **Asset2d** ($a), $f.id=$a.category.id

Il est important de noter que contrairement aux systèmes de partage de documents tels que Dropbox [2] ou Google Docs [3], notre système ne crée pas de copies locales pour définir la hiérarchie de répertoires. Celle-ci peut évoluer dynamiquement à chaque interaction puisqu'elle est définie par une règle qui est évaluée à chaque navigation dans son explorateur de fichier. Il en va de même pour les fichiers qui se retrouvent dynamiquement placés dans les répertoires par une requête dans la base de données.

4.3 Interrogation de partitions musicales

Les choix de représentation du contenu musical, que j'ai présenté dans la section 3.3, ont permis de définir une structure pour les collections de partitions et un modèle de données clairement défini pour la notation musicale. L'abstraction des formats musicaux standards va nous permettre de faciliter l'utilisation d'un langage d'interrogation dédié à la manipulation du contenu, sans se préoccuper du rendu visuel, du format ou des complexités induites par certaines notations. Plutôt que de définir un nouveau langage, le format XML nous permet d'utiliser XQuery. Toutefois, les manipulations musicales pourraient rester trop complexes pour intégrer et respecter l'intégrité du contenu musical. Pour ce faire, nous proposons l'algèbre "ScoreAlg" qui définit des opérateurs de haut niveau pour la manipulation de la notation musicale. Ces opérateurs seront utilisés comme fonctions dans les requêtes XQuery.

2. http ://dropbox.com
3. https ://docs.google.com

4.3.1 Une algèbre pour les partitions

L'algèbre "ScoreAlg" consiste en un ensemble d'opérateurs structurels prenant une (ou plusieurs) partition en entrée et en produit une nouvelle (forme close). De plus, notre algèbre inclut la possibilité de faire appel à des fonctions externes pour des manipulations spécifiques au contenu musical. Cette possibilité permet d'étendre la richesse du langage, du moment que la contrainte de clôture est respectée.

Fonctions. Une *fonction temporelle* est une correspondance de $\mathcal{T}$ vers $\mathcal{T}$. Nous ne considérons ici que les fonctions linéaires $\tau_{m,n}, m \neq 0$ de la forme : $\tau_{m,n}(t) = mt + n$. Nous pouvons citer deux sous-classes de fonctions temporelles, les fonctions de *déformations temporelles* : $warp_m : \mathcal{T} \to \mathcal{T}$, $t \mapsto mt$; et de *décalages temporels* : $shift_n : \mathcal{T} \to \mathcal{T}$, $t \mapsto t + n$. Ils modifient la durée ou décalent le moment des événements de $\mathcal{T}$.

Une *fonction de domaine* fait la correspondance de valeurs d'un espace multidimensionnel d'événements $\mathcal{E}(\mathbf{dom}_1) \times \cdots \times \mathcal{E}(\mathbf{dom}_n)$ vers une valeur simple d'un domaine d'événements $\mathcal{E}(\mathbf{dom}_o)$. Je présenterai l'interprétation de ce type de fonctions par la suite.

Les *fonctions principales* de l'algèbre comportent (i) les fonctions temporelles linéaires et (ii) les opérateurs internes sur les domaines de valeur. Voici quelques exemples d'application de fonctions principales sur l'événement **dsound** $e = D5^{20}_{12}$:

— $\tau_{2,0}(e) = D5^{40}_{24}$ (déformation temporelle) ;
— $\tau_{0,5}(e) = D5^{25}_{17}$ (décalage temporel) ;
— $\updownarrow(e, 2) = E5^{25}_{17}$ (transposition).

ScoreAlg $(\oplus, \mu, \sigma, \circ, \texttt{map})$ L'algèbre ScoreAlg comporte cinq opérateurs pour effectuer respectivement la *synchronisation* (combinaison de partitions), la *projection* (extraction de voix), la *sélection* (filtre sur les événements), la *fusion* (fusion de voix *"fusion-compatibles"* - définition 3.3.2) et le *map* permettant d'appliquer des fonctions externes. Chaque opérateur peut prendre une ou deux partitions en entrée pour en produire une nouvelle.

Définition 4.3.1 (Synchronisation – $\oplus$) **:** Si S_1 et S_2 sont deux partitions, alors $S_1 \oplus S_2$ est une partition définie par :

$$[S_1 \oplus S_2](t) = (S_1(t), S_2(t)), \forall t \in \mathcal{T}.$$

Définition 4.3.2 (Projection – μ) **:** Si S est une partition de type $[v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$, alors $\mu_{v_{i_1},\cdots,v_{i_m}}(S), \forall i_j, j \in [1, n]$ est une partition définie par :

$$[\mu_{v_{i_1},\cdots,v_{i_m}}(S)](t) = (S.v_{i_1}(t), \cdots S.v_{i_m}(t))$$

Définition 4.3.3 (Sélection – σ) **:** Si S est une partition de type $T = [v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$ et F une expression booléenne sur $\mathcal{T}, \mathbf{dom}_1, \cdots, \mathbf{dom}_n$, alors $\sigma_F(S)$ est de type T telle que chaque voix $S.v_i$:

$$[\sigma_F(S)].v_i(t) = \begin{cases} S.v_i(t), si\ F(S.v_i(t)) = \text{true} \\ \bot, \qquad\qquad\qquad sinon \end{cases}$$

Définition 4.3.4 (Fusion – $\circ$) **:** Si S_1 et S_2 sont deux partitions de type $T = [v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$, telles que $\mathcal{P}(S_1.v_i)$ est *"fusion-compatible"* avec $\mathcal{P}(S_2.v_i) = \emptyset, \forall i \in [1, n]$, alors $S_1 \circ S_2$ est de type T définie par :

$$[S_1 \circ S_2].v_i(t) = \begin{cases} S_1.v_i(t), si\ S_1.v_i(t) \neq \bot, S_2.v_i(t) = \bot \\ S_2.v_i(t), si\ S_1.v_i(t) = \bot, S_2.v_i(t) \neq \bot \\ \bot, \qquad si\ S_1.v_i(t) = S_2.v_i(t) = \bot \\ (S_1.v_i(t) \Xi S_2.v_i(t)), \qquad sinon \end{cases}$$

Exemple 4.3.1 : Les expressions algébriques de la page suivante illustrent les opérateurs sur les séries temporelles de types vScore présentées dans la figure 3.4 :

Définition 4.3.5 (Transformation – map) **:** Si S est une partition de type $[v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$, et f une fonction $\Pi_i^n \mathbf{dom}_i \to \mathbf{dom}_o$ alors $\texttt{map}_{a:f}(S)$ est une partition de type $[a : \mathbf{dom}_o]$ définie par :

$$[\texttt{map}_{a:f}(S)].a(t) = f(S.v_1(t), \cdots S.v_n)(t), \forall t \in \mathcal{T}$$

Synchronisation de S_{sopr} et S_{lyrics} :

$$[S_{sopr} \oplus S_{lyrics}](t) = \begin{cases} (r_0^{12}, r_0^{12}), & t \in [0,12[\\ (D5_{12}^{20}, \text{Ah}_{12}^{20}), & t \in [12,20[\\ (r_{20}^{21}, r_{20}^{21}), & t \in [20,21[\\ (E5_{21}^{22}, \text{que}_{21}^{22}), & t \in [21,22[\\ (F5_{22}^{23}, \text{je}_{22}^{23}), & t \in [22,23[\\ (D5_{24}^{28}, \text{sens}_{24}^{32}), & t \in [24,28[\\ (C5is_{28}^{32}, \text{sens}_{24}^{32}), & t \in [28,32[\\ (r_{32}^{34}, r_{32}^{34}), & t \in [32,34[\\ (A4_{34}^{36}, \text{d'in}_{34}^{36}), & t \in [34,36[\end{cases}$$

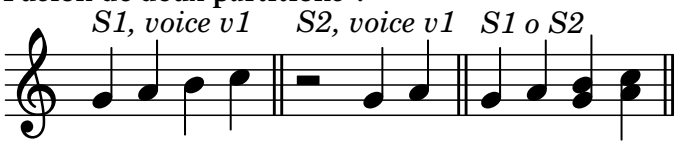
Projection de la voix lyrique S_{lyrics} (3^{eme} mesures) :

$$\mu_{sopr,bass}(S(t)) = \begin{cases} (D5_{12}^{20}, <B3es, D4>_{12}^{16}), & t \in [12,16[\\ (D5_{12}^{20}, A3_{16}^{20}), & t \in [16,20[\\ (r_{20}^{22}, G3_{20}^{24}), & t \in [20,22[\\ (E5_{22}^{23}, G3_{20}^{24}), & t \in [22,23[\\ (F5_{23}^{24}, G3_{20}^{24}), & t \in [23,24[\end{cases}$$

Sélection des mesures 12 à 24 :

$$[\sigma_{t \in [12,24[}(S_{sopr})](t) = \begin{cases} D5_{12}^{20}, & t \in [12,20[\\ r_{20}^{22}, & t \in [20,22[\\ E5_{22}^{23}, & t \in [22,23[\\ F5_{23}^{24}, & t \in [23,24[\end{cases}$$

Fusion de deux partitions :



L'opérateur map permet d'effectuer des transformations intéressantes telles que :

— Décaler la voix d'une partition de n unités temporelles vers la droite : $\text{map}_{r:\tau_{0,n}}(S)$.

— Transposer la voix d'une partition en montant de 5 demi-tons (unités **dsound**), avec : $\text{map}_{t:\updownarrow 5}(S)$.

— Calculer la séquence d'intervalles pour une partition contenant deux voix : $\text{map}_{i::-:}(S_{duo})$.

— Ou encore, appliquer une fonction utilisateur comme produire la séquence des durées (série temporelle d'entiers) pour une voix d'une partition S, avec : $\text{map}_{d:duration()}(S)$.

Expressions algébriques. Une expression est définie de manière classique puisque le résultat d'un opérateur reste une instance du modèle (vScore), donc la composition d'opérateurs permet de définir des expressions complexes :

— Si S est une partition, alors S est une expression.

— Si E_1 et E_2 sont deux expressions, alors $E_1 \oplus E_2$, $\text{map}_{a:f}(E_1)$, $E_1 \circ E_2$, $\sigma_F(E_1)$, $\mu_{v_{i_1},\cdots,v_{i_m}}(E)$ sont des expressions, si les conditions d'entrées des opérateurs sont respectées.

Opérateurs relationnels étendus. Afin de manipuler les collections d'opus (exemple du Quartet 3.3.3), il nous faut étendre l'algèbre relationnelle. Hormis les opérateurs usuels tels que la sélection σ, le produit cartésien $\times$, l'union $\cup$ et la différence $-$, nous étendons l'opérateur de projection, Π.

Définition 4.3.6 (Projection étendue, Π)**:** Soit E une expression algébrique valide ayant pour schéma $(t_1,\cdots,t_m)$. Si $e_1,\ldots,e_q$ sont des expressions ScoreAlg valides, alors : $\Pi_{[e_1(t_1),\cdots,e_q(t_q)]}(E), q \leq m$, est une expression algébrique relationnelle valide.

Sa sémantique sur une instance I est définie par :

$$\Pi_{[e_1(t_1),\cdots,e_q(t_q)]}(E) = < e_1(r.t_1),\cdots,e_q(r.t_q) > | r \in E(I)$$

Exemple 4.3.2 : Nous souhaitons extraire le titre et deux voix d'une collection de quartets de *Joseph Haydn*. La première voix est produite par la synchronisation du premier violon avec le violoncelle et la seconde voix par la synchronisation du second violon avec l'alto :

$$\Pi_{\text{title,v1}\oplus\text{cello,v2}\oplus\text{alto}}(\sigma_{\text{composer='Haydn'}}(\text{Quartet}))$$

Clôture et expressivité. La propriété suivante est déduite de sa définition.

Théorème 4.3.1 (Stabilité)**:** Soit S une partition et E une expression, alors $E(S)$ est une partition.

L'algèbre ScoreAlg est également complète par rapport à la génération et la transformation de partitions. Cette propriété de complétude, relative à la puissance d'expression de l'algèbre, montre sa capacité à définir l'ensemble de l'espace des partitions musicales (dans le contexte de la notation musicale).
Théorème 4.3.2 (Complétude)**:** Soit S_1 et S_2 deux partitions. Alors, il existe une expression E composée d'opérateurs de ScoreAlg, telle que $S_1 = E(S_2)$.

Preuve. Soit une partition simple en entrée S_a, avec une seule voix et un seul événement $a_{t_1}^{t_2}$. Alors :
- En appliquant les *fonctions principales* (*décalage*, *déformation* et *transformation*) et l'opérateur map, nous pouvons obtenir n'importe quelle autre voix avec un événement $b_{t_1}^{t_2}$.
- En fusionnant ces deux partitions avec l'opérateur $\circ$, nous obtenons n'importe quelle *voix*, y compris les événements complexes tels que les accords.
- En synchronisant les voix générées avec l'opérateur $\oplus$, n'importe quelle *partition* peut être générée.

Ainsi, E_l est une expression telle que $S_1 = E_l(S_a)$. Inversement, il est possible de montrer que de n'importe quelle partition S, nous pouvons produire une expression avec la sélection (σ) et la projection (μ) pour obtenir un seul événement. Alors, il existe E_q telle que $E_q(S_2) = S_a$. Pour conclure, $S_1 = E_l(E_q(S_2))$. □

4.3.2 XQuery + ScoreAlg = requêtes musicales

Les opérateurs de l'algèbre ScoreAlg ne suffisent pas pour manipuler simplement les partitions, il nous faut un langage pour exprimer n'importe quelle manipulation. En couplant XQuery avec ScoreAlg, cela permet de chercher, structurer, extraire et transformer les partitions codées en XML. Les exemples de requêtes XQuery ci-dessous présentent quelques fonctionnalités, y compris l'utilisation de fonctions utilisateurs, et reposent simplement sur notre modèle de données hiérarchique (opus, partitions, voix). Les expressions ScoreAlg sont utilisées comme fonctions (d'autres exemples sont présentés dans [FSRT16d]). Les problèmes d'implémentation du modèle et des opérateurs sont présentés dans la section 5.3.
Exemple 4.3.3 : Ce premier exemple illustre l'utilisation des opérateurs ScoreAlg : *sélection* des mesures 1 à 5 (select) et *projection* sur le soprano (XPath).

```
for $o in collection("Chorals")/opus
let $incipit := scoreql:select ($o/score/soprano, "measure() in [1,5]")
return <result>{$o/title}<incipit>{scoreql:eval($incipit)}</incipit></result>
```

Cette requête montre la manière de définir des variables et de produire de nouvelles partitions.

Exemple 4.3.4 : Cette requête illustre une fonction utilisateur "*highest*" qui sélectionne la plus haute note de la partition pour récupérer les chorales ayant une note supérieure à F5. Puis des transformations (map) sont appliquées aux voix soprano et basse pour descendre de 2 demi-tons. Le tout est ensuite synchronisé.

```
for $o in collection("Chorals")/opus
where scoreql:highest($o/score/soprano) > scoreql:frequency("F5")
let $transpS := scoreql:map ($o/score/soprano, "transpose (-2)")
let $transpB := scoreql:map ($o/score/bass, "transpose (-2)")
return   scoreql:eval (scoreql:sync ($transpS, $transpB))
```

Cette requête montre la manière de dériver du contenu et de recomposer des voix, mais aussi l'utilisation de fonctions utilisateurs.

4.4 Conclusion

Dans ce chapitre, nous avons vu comment chaque type de données pouvait être manipulé, et la manière d'interroger ces collections avec des langages dédiés. Nous pouvons noter que chacun d'entre eux repose sur le modèle de données et en exploite les propriétés. Pour résumer :
- *Les flux de données XML* sont manipulés à l'aide d'opérateurs continus produisant de nouveaux flux. L'essence de l'algèbre repose sur une évaluation au fil de l'eau, sans jamais bloquer le flux. L'astuce

est d'intégrer la notion d'annotations à travers des jointures spécifiques ; Le défi est maintenant d'optimiser l'évaluation multi opérateurs d'un ensemble de publications appelé *graphe de requêtes*.

— *Les flux de données textuelles* sont filtrés par des requêtes basées sur les mots-clés. Le processus de correspondance large est amélioré grâce à des correspondances partielles avec une pondération dédiée, ainsi qu'une étape de filtrage par nouveauté et diversité. La simplicité de ce processus dissimule une forte complexité dans l'optimisation du traitement en continu de centaines de millions de souscriptions. En effet, il va nous falloir maintenir l'historique de chaque souscription, calculer les distances et les similarités des items du flux et l'effectuer sur l'ensemble de l'historique.

— *Les requêtes Datalog* favorisent la définition de fait sur les données, devant être valides à chaque instant. Ce concept rend le système évolutif et stable, mais nécessite quelques restrictions pour éviter tout risque de règles non saines. L'étape suivante repose sur la manière de matérialiser les règles *actives* et d'évaluer efficacement à la volée les règles *dynamiques* sur des assets.

— *Pour être interrogées, les partitions de musique* nécessitent une vision simplifiée. Grâce à une algèbre dédiée aux séries temporelles musicales et à des objets adaptés, le langage XQuery peut alors être utilisé de manière simple. Ainsi, sous forme d'appels de fonctions, la complexité des manipulations musicales est cachée à l'utilisateur. Toutefois, le principal problème réside dans l'interrogation de documents XML stockés dans le format natif, ne correspondant pas au modèle de données spécifié.

J'ai pu durant ces travaux utiliser ces différents types de manipulation de données. Ils ne couvrent pas toute la richesse de possibilités offertes par d'autres modèles de données. Toutefois, il est important de noter qu'à chacun de ces modèles est dédié un ensemble de manipulations. Nous pouvons distinguer trois familles de manipulations parmi celles que j'ai présentées :

(1) Les **opérateurs algébriques** formalisant les notions de transformations. Le modèle de données influe sur le type d'opérateur et les propriétés à respecter pour la composition.

(2) La **manipulation d'espaces vectoriels** correspondant à un opérateur unique dont les propriétés reposent sur le calcul de pertinence. Elle a la particularité de considérer chaque donnée dans un espace de données pour en extraire du sens pour l'utilisateur.

(3) Les **règles déductives** décrivent formellement les propriétés que chaque donnée doit respecter. Les prédicats définissent clairement les contraintes et transformations sur les données.

Ces manipulations respectent à chaque fois la propriété fondamentale qu'est la forme close. C'est à dire qu'elles permettent de produire de nouvelles données dont les propriétés respectent le modèle de données manipulé. Le but *in-fine* est de permettre la composition d'opérations pour augmenter l'expressivité des manipulations.

Cette expressivité est facilitée par un langage d'interrogation. Son but est d'abstraire l'aspect opérationnel ou formel des manipulations. Le choix de ce langage est dirigé par les besoins de l'utilisateur final (simplifier les manipulations métiers), ainsi que respecter la logique du modèle de données :

— Les langages déclaratifs tels que `RoSeS` et `ScoreQL` illustrent cette idée de simplification des manipulations tout en préservant la logique de flux ou de partitions de musiques.

— Le langage naturel à base de mots clés pour filtrer les flux d'items quant à lui repose sur le fait que le modèle vectoriel et le langage ne font qu'un, facilitant l'expression des requêtes.

— Les règles déductives pourraient quant à elles être abstraites par un langage de plus au niveau pour définir des collections dédiées. Toutefois, un langage a bien été proposé pour les deux types de collections "*answer*" et "*File/Folder*" reposant sur les interactions d'une IHM naturelle pour exprimer les "requêtes".

Le modèle de données, les manipulations associées et le langage de requêtes forment le socle même de l'interrogation d'une base de données. Nous allons voir dans le chapitre suivant, la complexité des choix d'implémentation et d'optimisation liés aux choix effectués précédemment.

Chapitre 5
Intégration et Optimisation de requêtes

Comme nous avons pu le voir dans le chapitre précédent, le langage d'interrogation a pour but de dissocier ce que nous cherchons, de comment nous le cherchons. En effet, l'abstraction des traitements offerte par le langage permet au système de choisir comment, ces traitements seront appliqués lors de l'exécution de la requête. L'optimisation de requêtes dans une base de données est l'ensemble des améliorations de manipulations sur les données ayant pour but d'en réduire le temps d'évaluation.

Pour ce faire, différentes techniques sont possibles telles que les index, le stockage en mémoire centrale, l'organisation physique des données, la mutualisation des ressources, le *clustering*... Toutefois, chaque type d'optimisation n'est pas forcément compatible avec tous modèles de données ; indexation de valeurs scalaires *vs* texte, requêtes directes *vs* continues, cohérence *vs* disponibilité... Le contexte lié au modèle de données influe également sur les possibilités d'optimisation, comme le volume de données, la vélocité de mises à jour, interrogations concurrentes... Le choix du langage détermine aussi l'orientation technique telle que l'implémentation d'opérateurs d'une algèbre, l'indexation, l'extraction de la pertinence, l'évaluation de fonctions, le traitement des événements... Ainsi, nous allons voir comment le modèle de données et le langage d'interrogation associé influent sur l'optimisation de requêtes.

Dans ce chapitre, je vais présenter la démarche d'intégration et d'optimisation des langages présentés dans le chapitre précédent. Je mettrais l'accent sur les choix techniques et leur impact sur le système. Les thèmes d'optimisation abordés se focalisent sur :

(1) **Les graphes de requêtes** : les requêtes RoSeS produisent des séquences d'opérations sur les flux de données. Il est indispensable d'étudier la factorisation de ces opérateurs lorsqu'ils partagent des traitements communs (section 5.1.1).

(2) **Le Pub/Sub avec des mots-clés** : la vérification de la correspondance de millions de souscriptions sur un flux de publications en temps réel implique la création d'un index dédié, je présenterai et comparerai trois approches distinctes d'indexation de publications (section 5.1.2). Ensuite, l'optimisation du calcul de la pertinence des notifications par nouveauté et diversité sera abordée avec une implémentation centralisée (section 5.1.3), mais également avec une version alternative dans un contexte distribué, permettant de s'intéresser au modèle physique de données et en particulier à la dénormalisation de schémas (section 5.1.4).

(3) **Les règles Datalog** : l'évaluation de règles Datalog intégrées à la base de données implique l'implémentation d'un mécanisme de déclencheurs dédiés, mais pour un gros volume de données, un contexte distribué sera alors plus adapté. De plus, la conception d'une vue système de fichiers pour les assets graphiques sera présentée (section 5.2).

(4) **Les manipulations de partitions musicales** : la différence entre le modèle de données et l'encodage des partitions musicales, ainsi que la complexité des opérations sur des séries temporelles font que l'implémentation de ces opérateurs est le véritable enjeu, plutôt que l'optimisation du traitement en lui-même sur un *corpora* musical (Section 5.3).

5.1 Optimisation d'un système Pub/Sub

La particularité des systèmes de Publication/Souscription est de stocker les requêtes pour évaluer les données à la volée. Ainsi, l'optimisation de ce type de système repose sur le fait d'évaluer des millions de souscriptions sur les données en continu. La première règle à appliquer est de factoriser au maximum les traitements, car la probabilité que deux requêtes partagent une opération en commun est forte.

5.1.1 Optimisation basée sur les graphes

L'évaluation de requêtes dans RoSeS consiste à traiter des publications avec un plan *multi requêtes*, composé de différents opérateurs (présentés dans la section 4.1.1). La gestion de flux d'items est gérée par une interconnexion des opérateurs à l'aide de files d'attentes et de fenêtres (opérateurs bloquants).

Opérateur	Taux de sortie	Mémoire	CPU
$\sigma_p(b)$	$sel(p) * R(b)$	$const$	$const * R(b)$
$\cup(b_1, ..., b_n)$	$\sum_{1 \leq i \leq n} R(b_i)$	0	0
$\bowtie_p (b, w)$	$sel(p) * R(b)$	$const$	$R(b) * S(w)$
$\omega_d(b)$	0	$S = const \vert d * R(b)$	$const$

Table 5.1 – Modèle de coût pour graphes multi requêtes

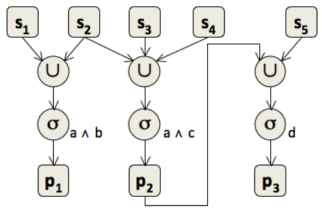


Figure 5.1 – Graphe multi requêtes

Ce plan multi requêtes est un graphe acyclique orienté $G(Q)$, comme illustré dans la figure 5.1. L'optimisation de ce graphe d'opérateurs repose sur un modèle de coût.

Modèle de coût pour un plan multi requêtes. Les requêtes de publication de flux sont traduites en expressions algébriques. Les requêtes partageant une même source peuvent alors se voir mutualiser des opérateurs (même filtre ou union). Chaque opérateur a été implémenté en *multithread* avec un modèle d'exécution pipeliné (classique pour le traitement de données en continu).

Chaque opérateur est accompagné d'un modèle de coût pour estimer la charge en mémoire, en CPU, mais plus particulièrement de la bande passante. L'estimation du coût doit refléter la nature même du flux de données. Nous avons simplifié le modèle présenté par *Cammert et al.* [CKSV08] en proposant un coût, sous forme de fonctions de taux de publication, $R(b)$ appliquées à un tampon d'entrée b (mais aussi une taille de fenêtre $S(w)$ pour les jointures).

La table 5.1 montre pour chaque opérateur son modèle de coût, nous pouvons constater qu'il repose essentiellement sur le taux de publication $R(b)$ en entrée. L'opérateur de jointure produit des items annotés par d'autres items provenant de la fenêtre S (s'il y a au moins une correspondance). L'opération de fenêtrage, quant à lui, transforme un flux sous forme de tampon dont la taille dépend du type de fenêtre (basé sur le temps ou le contenu des items).

Optimisation de graphes de requêtes. L'originalité de notre optimisation multi requêtes repose sur un modèle de coût applicable facilement à un graphe. Nous utilisons les méthodes traditionnelles de réécritures d'expressions algébriques : distributivité des sélections sur l'union, commutativité de la sélection avec la jointure et composition de sélections. Grâce à la définition de notre jointure et de leurs annotations, la commutativité avec celle-ci est possible et favorise l'optimisation.

Le processus d'optimisation est alors décomposé en deux phases : (a) la *normalisation* des expressions, en poussant les sélections sur les flux sources, et la distribution des jointures sur l'union, et (b) une *factorisation* des prédicats de sélection en fonction de leur coût.

Factorisation de prédicats. Le processus de factorisation produit pour chaque source, un plan d'exécution minimal composé d'opérateurs filtrants. Il est construit en deux étapes : 1) générer un *graphe de subsomption* de prédicats pour chaque source (les prédicats recouvrant d'autres prédicats de $P(s)$), 2) trouver un arbre de *Steiner* [CCC+98] correspondant au sous-arbre de coût minimal couvrant tous les prédicats.

La figure 5.2 montre à la fois le graphe de subsomption de la source s_2 et l'arbre de *Steiner* (en rouge). Chaque lien de subsomption est étiqueté par un poids provenant du modèle de coût (en fonction du taux de sortie). Il correspond au produit des fonctions de taux de publications combinés à la sélectivité de chaque prédicat présent sur le chemin (*ex.*, $cost(s_2 \rightarrow \sigma_a(s_2)) = rate(s_2) \cdot selectivity(a)$). Le coût du plan d'exécution, correspondant à un sous-graphe, est la somme de tous les poids des arcs le composant. Le choix du plan d'exécution minimal correspond au problème d'arbre de *Steiner* qui est `NP-complet`.

Afin de chercher le bon candidat, nous proposons une première heuristique qui exploite la propriété particulière de filtrage des opérateurs dont le poids décroît de manière monotone au fur et à mesure de leurs combinaisons. Nous commençons par la *Bordure*, un ensemble de prédicats à factoriser (publications : $a \wedge b \wedge c$, $b \wedge c \wedge d$, $a \wedge e$, $e \wedge f$). Chaque itération cherche donc à maximiser la subsomption de prédicats de la bordure, en d'autres termes, maximiser la factorisation. La bordure est alors échangée avec le nouveau nœud sélectionné et les prédicats restants. Le choix du candidat se fait en fonction de la sélectivité du prédicat et le nombre de prédicats qu'il peut factoriser. Ce choix peut se formaliser par une

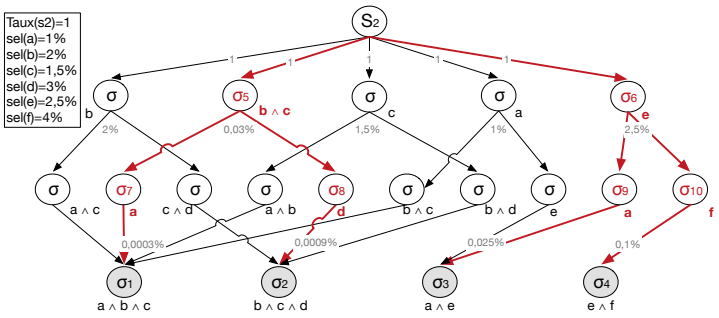


FIGURE 5.2 – Graphe de subsomption et Arbre de *Steiner*

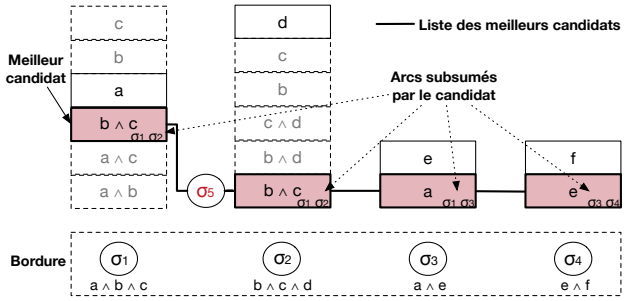


FIGURE 5.3 – VCB : Première itération

fonction de bénéfice apporté par x en étant fils de y dans l'arbre de *Steiner*, tel que :

$$benefit(x,y) = (k-1)\cdot selectivity(y) - k\cdot selectivity(x)$$

où k est le nombre de fils de y qui sont également subsumés par x. Ainsi, l'insertion de x implique de remplacer les arcs de y vers les k fils par ceux de x vers k. Cette fonction exprime la différence de coûts de l'arbre filtrant entre avant et après l'insertion de x. De fait, un bénéfice positif veut dire que l'insertion de x améliore la sélectivité globale, tandis qu'un score négatif suggère la suppression du nœud x.

Factorisation avec VCB. La première heuristique pose toutefois un problème lors de sa construction. Choisir le meilleur candidat à chaque étape implique que certains liens de subsomption ne sont jamais testés, la liste de candidats est calculée à chaque étape. Afin de réduire ce coût, nous avons défini une structure de données, appelée *VCB* (*Very Clever Border*), qui : 1) calcule à la volée le nombre minimal de liens de subsomptions nécessaires pour choisir le meilleur prédicats (meilleure factorisation), et 2) met à jour la liste de candidats de manière itérative. L'idée générale de cette heuristique est montrée dans la figure 5.3 (détaillée dans [CATV11b]). Pour résumer, on génère à l'avance la liste de subsomptions pour la *Bordure*, dont les nouvelles opérations sont triées par ordre de sélectivité.

5.1.2 Indexation de souscriptions

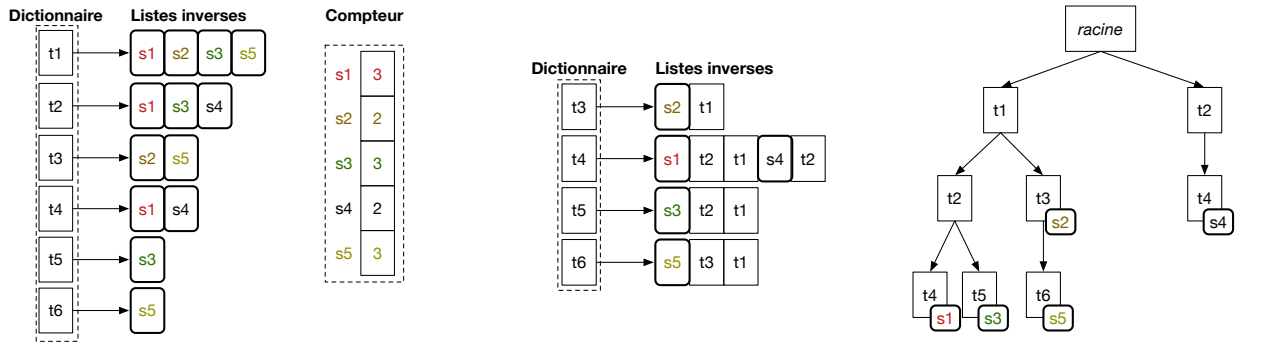
Le filtrage par requêtes larges (définition 4.1.6) dans un contexte Pub/Sub nécessite de calculer les correspondances d'items en temps réel sur des millions de souscriptions. Une structure d'index est nécessaire pour factoriser les traitements en communs, cela permettra de supprimer aussi vite que possible les souscriptions non pertinentes. Nous avons pu dans ce travail appliquer trois index différents pour factoriser trois types de factorisation, pour lesquels nous avons proposé un modèle analytique.

Le premier type de structure repose sur les bien connues listes inverses (IL) dont le rôle est de stocker la correspondance de chaque terme t_i vers les souscriptions qui le contient. Deux variantes de ces listes inverses sont possibles : **Count-based Inverted List** (CIL - liste avec compteur de termes) et le **Ranked-key Inverted List** (RIL - liste exploitant les rangs des termes). Le deuxième type de structure repose sur les arbres avec un **Regular Ordered Trie** (ROT - Trie ordonné) qui exploite les sous-ensembles de termes en communs entre les souscriptions, mais également une variante du trie : le *Patricia Trie* (POT).

Nous avons tout particulièrement mis l'accent sur une analyse approfondie de ces trois types d'index, aussi bien en termes de complexité qu'expérimentalement, dans un contexte de flux continu.

Count-based Inverted List (CIL). Cette structure de données est essentiellement un dictionnaire de termes, tels que $t_i \in \mathcal{V}_S$, et dont les valeurs présentes dans les listes ($Postings(t_i)$) correspondent aux termes des souscriptions. La ressemblance avec les techniques de RI s'arrête aux besoins des requêtes larges. En effet, une structure supplémentaire est nécessaire, un compteur pour chaque souscription, afin de mémoriser le nombre total de termes à trouver pour chacune. Cette structure fait donc correspondre, pour chaque souscription $s \in \mathcal{S}$, le nombre de termes qu'il reste à trouver dans l'item pour le notifier.

Le processus d'évaluation des correspondances doit : 1) initialiser ce compteur (copie d'un tableau), 2) accéder pour chaque terme $t_i \in I$ à la liste inverse $Postings(t_i)$, puis 3) pour chaque $s \in Postings(t_i)$ la



(a) Count-based Inverted List (CIL) (b) Ranked-key Inverted List (RIL) (c) Regular Ordered Trie (ROT)

FIGURE 5.4 – Subscription Indexes

valeur correspondante du compteur est décrémentée. Dès qu'une valeur du compteur atteint zéro, tous les termes de la souscription sont présents dans l'item, et ce dernier est notifié pour cette souscription.

La figure 5.4 a) montre la structure de l'index et de son compteur. Prenons l'exemple d'un item $I = \{t_2, t_4, t_6\}$, la valeur du compteur, après avoir accédé aux termes t_2 et t_4, est égale à $\{S_1 : 1, S_2 : 2, S_3 : 2, S_4 : 0, S_5 : 2\}$, une correspondance est alors obtenue pour la souscription S_4.

Ranked-key Inverted List (RIL). Dans cet index, une souscription n'est stockée que dans une seule liste inverse, celle dont le terme est le moins fréquent parmi tous ces termes. Ce terme est appelé la *clé* de la souscription. La liste contient donc l'identifiant de la souscription, et les autres termes qui la constituent. Grâce à cette structure, il n'est plus nécessaire de gérer un compteur, car la *clé* oriente les correspondances. Les listes de termes fréquents sont donc réduites comparées à *CIL* et les souscriptions sont distribuées sur l'ensemble des termes, principalement sur les termes de fréquence médiane.

Pour procéder aux correspondances d'un item I sur le *RIL* : 1) les termes sont triés par rang, 2) pour chaque terme $t_i \in I$, la liste inverse $Postings(t_i)$ est accédée, 3) pour chaque souscription, si les termes restants sont contenus dans l'item (parmi les termes de rang inférieur), une notification est produite.

Notre item $I = \{t_2, t_4, t_6\}$ traité par l'index de la figure 5.4 b) commence par la liste $Posting(t_6)$; t_1 et t_3 ne sont pas présents dans l'item, donc aucune notification n'est produite. Puis la liste $Posting(t_4)$ est parcourue, s_1 n'est pas validée, par contre s_4 qui contient le terme t_2 correspond au contenu de l'item est alors notifiée. $Posting(t_2)$ ne contient aucune souscription (terme trop fréquent) et n'est pas parcourue.

Regular Ordered Trie (ROT). Organisé en hiérarchie de termes, le ROT est une alternative aux listes inverses. Il exploite la factorisation des termes communs des souscriptions. Nous obtenons un Trie dont les nœuds contiennent un terme et les souscriptions dont le chemin de la racine vers ce nœud constitue les termes de ces souscriptions. Lorsque deux souscriptions partagent un sous-ensemble de termes k, un chemin commun de longueur k sera constitué, suivi par deux chemins distincts avec les termes restant.

Les souscriptions sont ainsi stockées à un seul endroit, et aucun compteur n'est nécessaire. Par rapport aux structures de Tries classiques pour des séquences de termes [Knu73], deux caractéristiques diffèrent : (i) il n'y a pas de répétition de termes (une souscription est un *ensemble* de termes), (ii) les termes respectent un *ordre total*. Cette structure correspond donc à un *Regular Ordered Trie* (*ROT*) étudié dans un contexte différent [YGM94], et plus récemment en *Data Mining* [MK07].

La correspondance d'un item I : 1) commence par un tri des termes, 2) parcourt pour chaque mot de I les chemins dans le Trie, 3) à chaque nœud visité, les souscriptions associées sont alors notifiées.

Notre item I=$\{t_2, t_4, t_6\}$ recherche dans l'arbre de t_2 (figure 5.4 c), trouve le terme t_4 qui notifie la souscription s_4. Les termes t_4 et t_6 n'ayant pas d'arbre associé, le processus se termine.

Modèle analytique. Afin d'estimer pour chaque structure, la place mémoire, le temps de construction et d'évaluation des correspondances, nous proposons un modèle analytique basé sur la prédiction du nombre de nœuds (construits ou visités). Les paramètres et notations sont résumés dans le tableau 5.2.

$\|\mathcal{S}\|$	Nombre total de souscriptions
$\|\mathcal{V}_I\|, \|\mathcal{V}_S\|$	Taille du vocabulaire des items (resp. souscriptions)
$\|s\|_{avg}, \|s\|_{max}$	Taille moyenne des souscriptions (resp. taille maximum)
$\|I\|$	Taille moyenne d'un item
$P(t_i)$	Distribution des fréquences des termes de $\mathcal{V}_I$
$\theta(k)$	Probabilité qu'une souscription atteigne la taille k
σ_i	Probabilité qu'un terme obtienne un rang $\leq i$
$w(c), w(v)$	Taille d'une entrée du compteur (resp. dictionnaire)
$w(p), w(n)$	Taille d'une entrée de la liste inverse (resp. Trie)

Table 5.2 – Paramètres et notations du modèle analytique

Notons que $|s|_{avg}=\Sigma_{k=1}^{|s|_{max}}\theta(k)\times k$ défini la taille moyenne d'une souscription où $\theta(k)$ est la probabilité qu'une souscription ait la taille k (avec $k \in [1, |s|_{max}]$). $P(t_i)$ correspond à la fréquence des occurrences du terme $t_j \in \mathcal{V}_I$ qui suppose que le choix d'un terme $t_j \in s$ est indépendant du choix du terme $t_m \in s$. Ainsi, le nombre de termes du vocabulaire $\mathcal{V}_S$ est égal à : $|\mathcal{V}_S| = \sum_{i=1}^{|\mathcal{V}_I|} Pr(t_i \in \mathcal{S})$. De même, la probabilité que t_i soit présent dans au moins une souscription de $\mathcal{S}$ est définie par :

$$Pr(t_j \in \mathcal{S}) = 1 - Pr(t_i \notin \mathcal{S}) = 1 - (\sum_{k=1}^{|s|_{max}} \theta(k) \times (1 - P(t_i))^k)^{|\mathcal{S}|}$$

Je résumerai par la suite le modèle analytique (détaillée dans [HKC+16]) pour lequel je mettrai l'accent sur notre principale contribution qu'est le *ROT*.

Taille en mémoire. La table 5.3 donne les besoins en mémoire pour chaque structure.

Size(CIL)	=	Size(Dictionnary)	+ Size(Counter)	+ Size(Postings)
	=	$\|\mathcal{V}_S\| \times w(v)$	$+ \|\mathcal{S}\| \times w(c)$	$+ \|\mathcal{S}\| \times \|s\|_{avg} \times w(p)$
Size(RIL)	=	Size(Dictionary)		+ Size(Postings)
	=	$\sum_{i=1}^{\|\mathcal{V}_I\|} 1 - (1 - Post(s, t_i))^{\|\mathcal{S}\|} \times w(v)$		$+ \|\mathcal{S}\| \times \|s\|_{avg} \times w(p)$
Size(ROT)	=	$E(\lambda, 0) \times w(n)$		

Table 5.3 – Place mémoire requise pour chaque structure

La taille de **CIL** est décomposée en un compteur, un dictionnaire et les listes inverses. La taille de ces listes ($|\mathcal{S}| \times |s|_{avg}$) dépend du nombre de souscriptions et de leur taille moyenne.

La taille de **RIL** est décomposée en un dictionnaire et les listes inverses. Celles-ci dépendent de la probabilité que le terme correspondant soit le moins fréquent d'au moins une souscription de S. Ainsi, la souscription s est dans la liste $Postings(t_i)$ ssi $t_i \in s$ et qu'il n'y ai aucun terme $t_h \in s$ tel que $h < i$:

$$Post(s, t_i) = \sum_{k=1}^{|s|_{max}} \theta(k) \times k \times P(t_i) \times (\sigma_{i-1})^{k-1}$$

Concernant le Trie, il est nécessaire de compter le nombre de nœuds du **ROT**. Notre modèle analytique prend en compte la distribution des termes et la taille des souscriptions. Toutefois, du fait que le modèle soit récursif, il n'a pas de forme close, surtout pour de grands vocabulaires et items. Heureusement, nous nous focalisons sur les souscriptions dont le vocabulaire et la taille sont bornés.

Soit $\mathcal{P}$ le chemin de la racine vers le nœud du terme t_i dont le label (dénoté par Λ) est égal à : i) λ lorsque le chemin est vide, ii) $\Lambda \bullet i$ pour lequel Λ réfère au label du nœud père t_h qui a pour fils t_i ; le chemin $\mathcal{P}$ se termine donc par les termes t_h, t_i (le rang de t_h est inférieur à t_i).

Soit la souscription s, il existe alors un chemin dans le Trie dont la séquence des termes de s est ordonnée par leur rangs. Il est à noter qu'il existe $\binom{k}{k-|P|}$ souscriptions partageant le même chemin de label Λ. Ainsi, $Q(\Lambda, i)$ donne la probabilité que le nœud d'adresse $\Lambda \bullet i$ appartienne à une souscription s.

Lemme 5.1.1 : $Q(\Lambda,i) = \sum_{k=|\mathcal{P}|+1}^{|s|_{max}} \theta(k) \times Q(\Lambda,i,k)$ où $Q(\Lambda,i,k)$ donne la probabilité qu'un nœud labelisé par $\Lambda \bullet i$ appartienne à une souscription de taille k, qui est égale à :

$$Q(\Lambda,i,k) \quad = \quad \binom{k}{k-|\mathcal{P}|} \prod_{m \in \mathcal{P}} P(t_m) \times P(t_i) \times (1-\sigma_i)^{(k-|\mathcal{P}|-1)} \tag{5.1}$$

Preuve 5.1.1 : La probabilité que t_i appartienne à s à l'adresse $\Lambda \bullet i$ est égale à :

$$\Pi_{m \in \mathcal{P}} P(t_m) \times P(t_i) \times (1-\sigma_i)^{(k-|\mathcal{P}|-1)}$$

À partir du moment où les $k-|\mathcal{P}|-1$ nœuds restant de s doivent être choisis dans $\mathcal{V}_I - \{t_1,\ldots,t_i\}$ (avec pour probabilité σ_i), et qu'il y a $\binom{k}{k-|\mathcal{P}|}$ possibilités de souscriptions, nous pouvons en déduire l'équation 5.1.

Maintenant, soit $P(\Lambda,i)$ la probabilité qu'un nœud n avec pour label $\Lambda \bullet i$ existe dans *au moins une* souscription. Nous dirons alors que le nœud n est occupé avec une probabilité de $P(\Lambda,i)$. Ainsi, un nœud est inoccupé si l'adresse $\Lambda \bullet i$ n'est présente dans aucune des $|\mathcal{S}|$ souscriptions. Nous avons alors :

$$P(\Lambda,i) = 1 - (1 - Q(\Lambda,i))^{|\mathcal{S}|} \tag{5.2}$$

Ensuite, soit $E(\Lambda,i)$ le nombre de nœuds attendus dans le sous Trie ayant pour préfixe $\Lambda \bullet i$.

Théorème 5.1.1 : $E(\Lambda,i) = \sum_{m=i+1}^{|\mathcal{V}_I|-|s|_{max}+|\mathcal{P}|} P(\Lambda \bullet i, m) \times [1 + E(\Lambda \bullet i, m)]$ avec $E(\Lambda,i) = 0$ si $|\mathcal{P}| > s$ ou si $i \geq |\mathcal{V}_I|$

En effet, si un nœud $\Lambda \bullet h \bullet i$ est occupé, la taille du sous Trie de racine $\Lambda \bullet h \bullet i$ est égale à $E(\Lambda \bullet h, i)$.
Finalement, nous pouvons estimer la taille de notre *ROT* grâce à la formule suivante :

$$Size(ROT) \quad = \quad E(\lambda,0) \times w(n)$$

Temps d'évaluation. La table 5.4 donne l'estimation du temps d'évaluation pour chaque index.

TimeMatch(CIL)	=	Time(Copy_counter)	+ Time(Postings)
	=	$\lvert\mathcal{S}\rvert \times \tau_{copy}$	$+ \tau_{decr} \times \sum_{i=1}^{\lvert I\rvert} \frac{Pr(t_i \in \mathcal{S})}{\sum_{i=1}^{\lvert\mathcal{V}_S\rvert} Pr(t_i \in \mathcal{S})} \times \lvert\mathcal{S}\rvert \times \lvert s\rvert_{avg}$
TimeMatch(RIL)	=	Time(Sort)	+ Time(Postings)
	=	$\lvert I\rvert \times \log\lvert I\rvert$	$+ \lvert I\rvert \times \sum_{i=1}^{\lvert I\rvert} \lvert\mathcal{S}\rvert \times Posts(s,t_i) \times \tau_{chk}$
TimeMatch(ROT)	=	Time(Sort)	+ Time(Nodes)
	=	$\lvert I\rvert \times \log\lvert I\rvert$	$+ E(\lambda,0) \times \tau_n$

Table 5.4 – Temps d'évaluation des correspondances pour chaque structure

Une correspondance dans le **CIL** demande tout d'abord la copie du compteur, puis de parcourir la taille des listes inverses pour chaque terme de l'item de longueur $|I|$. La taille des listes inverses dépend de la somme des probabilités des termes de chaque souscription.

Pour le **RIL**, une correspondance demande le tri des termes de l'item, puis de parcourir pour chaque terme (décroissant) de l'item, les listes pour chaque inclusion de termes (τ_{chk}).

L'évaluation d'un item sur le **ROT** dépend du temps du tri des termes de l'item et de la somme des temps requis pour visiter les nœuds à partir de la racine. Le temps pour visiter un nœud est de τ_n.

Toutefois, l'ensemble des souscriptions obéi à une distribution particulière $Dist_k$ avec un vocabulaire dédié $\mathcal{V}$. De fait, le *ROT* résultant de cette restriction est dénoté par $T(S, Dist_k, \mathcal{V})$ et défini par :

Définition 5.1.1 (Restriction du Trie)**:** La restriction $T'(S, Dist_k, \mathcal{V}') = \Delta_{V'}(T(S, Dist_k, \mathcal{V}))$ est un sous Trie de T par le vocabulaire $\mathcal{V}' \subset \mathcal{V}$ avec une taille maximale de souscription identique $|s|_{max}$.

T' est donc élagué des termes non présents dans $\mathcal{V}'$. Il contient les souscriptions s dont les termes sont tous dans $\mathcal{V}'$, ainsi que les souscriptions dont le préfixe est défini dans $\mathcal{V}'$ mais sans le reste du chemin.

Le théorème 5.1.2 permet alors de prédire le nombre de nœuds visités pour un item sur un ensemble de souscriptions. L'ensemble des termes d'un item étant assez petit (cf. section 3.1.1), le nombre de nœuds visités sur la restriction du *ROT*, $T'(S, Dist_k, \mathcal{V}(I))$, au vocabulaire de cet item $\mathcal{V}(I)$ permet de déduire le temps de parcours du Trie. T' sera utilisé comme raccourci de $T'(S, Dist_k, \mathcal{V}(I))$.

Théorème 5.1.2: Le nombre estimé de nœuds visités pour effectuer la correspondance d'un item I sur le Trie T de souscriptions est égal au nombre de nœuds obtenu par la restriction $\Delta_{\mathcal{V}(I)}(T)$ où $\mathcal{V}(I)$ est le vocabulaire de l'item I. Nous pouvons alors exprimer le nombre de nœuds visités grâce à la formule 5.2 :

$$E(\lambda, i) = \sum_{m=i+1}^{|I|-|s|_{max}+1} P(\lambda \bullet i, m) \times (1 + E(\Lambda \bullet i, m))$$

Preuve 5.1.2: Le théorème 5.1.2 évalue le nombre moyen de nœuds visités. Soit n a des souscriptions, soit le sous Trie de racine n contient des souscriptions à vérifier. Un nœud n'a pas de fils s'il est de profondeur maximale ou si les fils n'appartiennent pas au vocabulaire $\mathcal{V}_I$. Il n'y a alors pas d'autres nœuds à visiter.

Évaluation des performances. Basé sur ce modèle analytique, nous avons alors étudié le comportement des indexes sur le gain en factorisation, en termes de nœuds abstraits dans l'arbre et les listes inverses. Bien que théoriquement idéal en factorisation, le *ROT* perd de son efficacité, car l'implémentation de ses nœuds requiert plus de ressources en mémoire. Les résultats ont montré que le *CIL* (vs. *ROT*) donne la plus petite (vs. grande) place en mémoire, alors que le *ROT* (vs *CIL*) offre les meilleurs (vs. pires) performances. Le *RIL* se place comme un bon compromis dont la mémoire s'approche du *CIL* et les performances légèrement supérieures au *ROT* (moins pour de grands nombres de souscriptions).

Pour résumer les caractéristiques que nous avons pu extraire de nos expériences :
i) La factorisation de nœuds est importante pour le *ROT* pour de petites souscriptions ; nous nous attendons à ce que les petites souscriptions reflètent la réalité des systèmes Pub/Sub, comme pour les requêtes Web [Vou10]. De plus, nous avons pu montrer que le taux de factorisation n'est que peu sensible à l'ordre des termes lors de la création du *ROT*.
ii) Bien que la distribution des occurrences des termes n'ait pas d'impact sur la taille des index, il en a une importante sur le temps d'évaluation. En particulier dans le cas où les items respecteraient la distribution des souscriptions, l'espace de recherche du *CIL* est très large comparé au *ROT* [HVT+11]. Ce dernier effectue trois ordres de grandeurs en moins de nœuds que le *CIL*. Pour des distributions différentes, le nombre de nœuds visités chute drastiquement.
iii) La taille des index grandie linéairement en fonction du vocabulaire. Par contre, pour de petits vocabulaires, *ROT* est plus efficace que *RIL* (un ordre de grandeur), mais pour de grands vocabulaires, *RIL* obtient une convergence du temps d'évaluation tandis que le *ROT* croit exponentiellement.
iv) Pour finir, la mémoire et le temps d'évaluation passent à l'échelle de manière linéaire pour les trois structures en fonction du nombre de souscriptions.

Correspondances partielles. Comme nous l'avions vu dans la section 4.1.2, les correspondances partielles sont utiles pour les souscriptions n'obtenant aucunes notifications. Nous avons donc étudié le moyen d'intégrer une contrainte moins forte sur les requêtes larges en intégrant un seuil κ pour les notifications dans chacun des trois index. L'approche simple consiste à produire pour chaque requête, l'ensemble des combinaisons de termes dont le score dépasse le seuil κ. Toutefois, la conséquence est une augmentation de la taille et des performances (comme vu ci-dessus), qui dans le pire des cas, obtiendra $2^{|s|} - 1$ possibilités. Bien sûr, la factorisation permet aussi de réduire cet effet dans le cas du *ROT*.

Une autre possibilité est d'étendre l'index *CIL* en y intégrant cette notion de correspondance partielle. Les structures *RIL* et *ROT* repose sur un ordre total imposant au moins un terme obligatoire, et ne peuvent

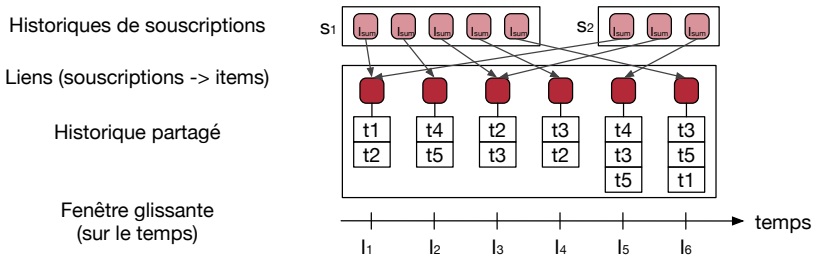


FIGURE 5.5 – Historique partagé pour le calcul de la nouveauté et de la diversité

prétendre à une telle extension. En effet, une correspondance partielle sans ce terme devient impossible. Le *CIL* est donc modifié en y intégrant un poids (TDV normalisé) pour chaque terme à l'intérieur des listes inverses, et le compteur devient un *compteur cumulatif*. Ainsi, pour chaque souscription s trouvée dans une liste inverse, le poids du terme est cumulé dans le compteur à la case correspondante. Si la somme cumulée dépasse le seuil κ, une notification est alors soulevée. avigation Nos expériences ont pu montrer que le *CIL* est bien plus performant que le *RIL* ou le *ROT* pour de grands nombre de souscriptions, du fait de la multiplication de la première solution.

5.1.3 Filtrage par Nouveauté et Diversité

Intégrer le filtrage par nouveauté et diversité dans un système Pub/Sub permet d'améliorer la qualité des notifications (section 4.1.2). Je présente ici une solution pour calculer efficacement les similarités et les distances entre les items des historiques H de millions de souscriptions en temps réel.

Historique partagé. À partir du moment où un item peut être partagé par plusieurs historiques de souscription H, nous devons éviter de garder l'ensemble des items. Si nous choisissons de garder les N derniers items de l'historique (fenêtre *item-based*) pour optimiser les ressources, cela ne prend pas en compte le taux de publication des items ; un fort taux va faire supprimer un item trop rapidement (et ne plus filtrer), et un taux faible risque de faire persister un item dans la fenêtre (trop filtrer). De fait, pour optimiser la place mémoire, nous avons opté pour un *historique partagé* $\mathcal{W}$ basé sur le temps (*time-based*), comme illustré dans la figure 5.5. C'est une fenêtre glissante qui contient tous les items notifiés au moins une fois sur une période de temps p. Les historiques de chaque souscription stockent alors une liste ordonnée de pointeurs sur les items correspondants dans $\mathcal{W}$.

Algorithme de filtrage. Filtrer un item I par nouveauté et diversité avec un grand nombre de souscriptions qui ont des items en commun soulève un réel défi d'optimisation. Pour permettre le passage à l'échelle, nous avons proposé de partager également le processus de filtrage entre toutes les souscriptions. L'algorithme (détaillé dans [HdMT15b]) va optimiser les calculs pour chaque item I' provenant des historiques de souscription H.

Le premier type d'optimisation va bénéficier des cooccurrences dans l'ensemble des historiques. Ainsi, les calculs $new(I,I')$ et $dist(I,I')$ ne seront calculés qu'une seule fois pour tous les calculs de nouveauté et de diversité (définitions 4.1.9 et 4.1.10). Le gain obtenu dépend du taux de cooccurrence des pairs d'items dans les historiques de souscription.

Le second type d'optimisation s'intéresse au calcul de la densité qui varie pour chaque item notifié sur un historique. Afin d'éviter une complexité quadratique pour calculer la somme de toutes les distances de H, l'algorithme va stocker les sommes de pairs de distances déjà calculées, dénommées par $I.sum$, pour chaque item de l'historique avec les items plus récents. Ainsi, la densité est simplifiée à une somme de $I.sum$. Grâce au fait que les items les plus anciens sont retirés plus tôt, aucune mise à jour n'est nécessaire lors de leur suppression. Ensuite, la comparaison de densités entre $D(H')$ et $D(H)$ peut être simplifiée, car $|H| = |H'|$ et $H' = H \cup \{I_n\} - \{I_o\}$. De la définition 4.1.10, le calcul de la diversité se résume par :

$$\frac{2 \times \sum_{I' \in H'} I'.sum}{|H'| \times (|H'|-1)} > \frac{2 \times \sum_{I' \in H} I'.sum}{|H| \times (|H|-1)} \Rightarrow I.sum > I_o.sum$$

```
{ "subID": 1,
  "terms": [103, 1710],
  "history": [
    {"itemID": 12, "sumDist": 0,
      "terms" : [103,162,543, "..."]},
    {"itemID" : 8, "sumDist" : 0.11689444,
      "terms" : [103, 314, 1527, "..."]} ]}
```

Figure 5.6 – Subscription-based model

```
{ "itemID": 1,
  "terms" : [78, 162, 208, 560, "..."],
  "subscriptions" : [
    {"subID" : 2736, "sumDist": [0.4418]},
    {"subID" : 3734, "sumDist": [0.4418]},
    {"subID" : 7762},
    {"subID" : 9261}] }
```

Figure 5.7 – Item-based model

Ainsi, la complexité de notre algorithme bénéficie de la cooccurrence des couples d'items pour la nouveauté et la diversité, ce qui permet d'obtenir une complexité linéaire.

Nos expériences ont permis de montrer que les filtrages par nouveauté et diversité étaient complémentaires. De plus, nous avons observé que le taux de filtrage dépend principalement du seuil de nouveauté et de la taille de la fenêtre (plage de temps), contrairement au filtrage par diversité. L'étude des performances du système a permis que le gain de temps fût obtenu essentiellement (97%) grâce à la factorisation des cooccurrences des termes. Pour finir, nous avons effectué une évaluation qualitative de notre système à l'aide d'un jeu de données réel et une validation par utilisateurs. La comparaison avec une approche top k traditionnelle a pu montrer que la contrainte de filtrage en temps réel (contrainte d'ordonnancement des filtres) avait un impact significatif sur la qualité du résultat. Cela nous a également permis de valider la pertinence du choix de la pondération des termes avec des TDV.

5.1.4 Pub/Sub dans un environnement distribué

Afin d'aller une étape plus loin dans nos travaux sur les systèmes Pub/Sub, nous avons étudié son intégration dans un contexte distribué et sa comparaison avec une optimisation centralisée [dMT18]. Nous avons fait le choix d'intégrer notre solution dans une base de données `MongoDB` [Ban12], dans laquelle nous avons dû adapter notre modèle physique de données, ainsi que distribuer le calcul des correspondances, de la nouveauté et de la diversité.

Cette nouvelle version de l'implémentation du modèle de données et de son langage de manipulation est un point intéressant de ce mémoire, car il permet de mettre en valeur le fait que les choix de modèle physique de données ont des conséquences sur la complexité d'implémentation et les performances.

Modèles physiques de données dénormalisés. Le principal problème d'une base de données distribuée est d'effectuer la jointure entre deux collections d'éléments. Dans notre contexte Pub/Sub, il y a une forte corrélation entre les items et les souscriptions. De fait, il est nécessaire de modéliser physiquement cette corrélation (dénormalisation), pour des raisons de performances. Toutefois, il existe deux possibilités de modélisations physiques : orientée "*souscriptions*" (stocker l'historique d'items) et orientée "*items*" (stocker les souscriptions notifiées). Les figures 5.6 et 5.7 illustrent ces choix pour lesquels nous retrouvons les termes, mais également de manière imbriquée, l'historique d'items pour l'un, et les souscriptions notifiées pour l'autre. Nous pouvons noter la présence d'une clé *sumDist* qui représente la valeur $I.sum$ présentée ci-dessus. J'ai volontairement retiré les valeurs de TDV pour simplifier le modèle.

Le modèle physique orienté "*souscriptions*" à l'avantage de pouvoir gérer naturellement les items liés à une souscription. Toutefois, les données d'un item seront dupliquées dans toutes les souscriptions (l'historique n'est pas partagé). Le principal problème de cette modélisation repose sur l'effet des mises à jour de cet historique de souscription. En effet, cette liste grandie au fur et à mesure des notifications, ce qui correspond à un problème critique connu dans les bases de données (équivalent au PCTFREE).

Le modèle physique orienté "*items*" résout ce problème en stockant l'item lorsqu'il est notifié (insertion) et ne nécessite aucune mise à jour ultérieure. Toutefois, le modèle implique une réécriture complète de l'algorithme de filtrage, car l'historique de chaque souscription est distribué sur l'ensemble des items, ce qui nécessite un regroupement des souscriptions (équivalent au "*group by*").

Stratégies de distribution. L'intégration du filtrage par nouveauté et diversité dépend du modèle physique choisi. Il a été implémenté en Map/Reduce, favorisant le passage à l'échelle horizontal en distribuant les données sur un ensemble de *shards* (serveurs). Le processus est décomposé en une phase Map dont le rôle est de filtrer les données en entrée, et la phase Reduce pour agréger les données produites par le *Map*. Nous avons intégré les fonctions de similarités, de distance et de densité.

Dans le cadre d'un modèle orienté "*souscriptions*", la nouveauté et la diversité du nouvel item sont naturellement testées dans le Map sur les items de l'historique. L'opération Reduce ne fait que récupérer les souscriptions notifiées à mettre à jour avec les données calculées.

Pour le modèle orienté "*items*", la phase Map est appliquée à chaque item dont il va pouvoir appliquer une seule fois les calculs de distance et de similarité en bénéficiant de la cooccurrence des items (les souscriptions concernées sont liées à cet item). La phase Reduce va agréger les items par souscription et calculer la diversité de l'historique recomposé. Le résultat produit la liste des souscriptions notifiées que l'on imbrique dans le nouvel item. Cette implémentation génère par contre de nombreuses communications réseau lors du regroupement (appelé Shuffle), c'est pourquoi nous avons groupé les items par l'identifiant de la souscription la plus "populaire" de l'historique.

Bien entendu, le modèle physique orienté "*items*" obtient des performances plus appropriées avec une convergence du temps de réponse, quel que soit le nombre de souscriptions. Le gain est principalement dû à la factorisation, même si la phase de Shuffle rallonge la convergence pour de nombreuses souscriptions. Les expériences montrent clairement que le modèle orienté "*souscriptions*" ne passe pas à l'échelle à cause des mises à jour imposant une redistribution des paquets de données (appelés *chunk*).

Par contre, en comparant l'implémentation centralisée à celle effectuée en milieu distribué, l'efficacité de la factorisation globale a un impact beaucoup plus fort sur le temps de calcul que la parallélisation des calculs. De fait, nous n'avons pu montrer expérimentalement le croisement théorique des courbes, avec un volume de plus de 500 millions de souscriptions sur plus de 30 serveurs, la distribution pourrait devenir intéressante. Nous pourrions appeler cela le seuil du "Big Data" dans ce contexte.

5.2 Le projet Polymathic

Le langage à base de règles présenté dans la section 4.2 permet de définir un système actif dirigé par les données. En effet, les règles sont déclenchées chaque fois qu'une donnée est mise à jour. Pour intégrer cela, nous avons besoin de : 1) définir le fonctionnement d'un système à base de déclencheurs passant à l'échelle, 2) vérifier si l'exécution des règles est saine, 3) matérialiser les données dérivées (pouvant déclencher de nouvelles règles) et 4) intégrer un bureau interactif avec la base de données.

5.2.1 Déclenchement des règles

L'architecture de Polymathic est illustrée dans la figure 5.8. L'originalité de notre approche repose sur la coordination entre le moteur de règles (*Rule engine*), la couche de persistance des données (*MongoDB*) et l'interface avec l'espace utilisateur (*FUSE*).

La couche de persistance est une base de données MongoDB pour laquelle un moniteur d'événements capture les événements qui seront empilés dans le moniteur pour vérifier le déclenchement des règles. Le moniteur va également vérifier le chaînage des opérations pour la détection de cycles. Le moteur de règles va maintenir le schéma et les contraintes du système Polymathic ; il gère les collections stockées dans MongoDB et l'interfaçage avec les fonctions spécifiques (cf. section 4.2.1). Le moteur doit également s'occuper de l'exécution des règles, aussi bien pour les recherches que pour la synchronisation de collections intentionnelles. Pour finir, la couche "FUSE" qui fait l'interface avec l'utilisateur publie les données nécessaires (Fichiers et Répertoires, cf. section 4.2.3) au Système de Fichier Virtuel (VFS).

Le moniteur d'événements. Pour des raisons de performances, nous avons choisi d'adopter une stratégie de *matérialisation distribuée* pour nos collections intentionnelles. Pour ce faire, une maintenance incrémentale du contenu des collections doit être effectuée afin de maintenir la cohérence avec les collections

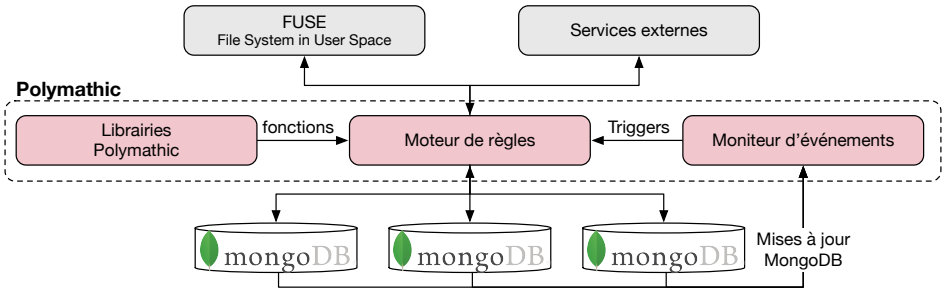


FIGURE 5.8 – Architecture simplifiée de Polymathic

extensionnelles. Comme MongoDB ne propose pas de système de *Triggers*, nous avons développé un moniteur temps réel qui capture tous les événements de la base, pouvant déclencher les règles définies.

Ainsi, nous avons exploité le fichier de *log* de MongoDB (appelé *oplog*) qui à l'origine est prévu pour gérer la réplication des données [Ban12]. Toutes les opérations de mises à jour y sont intégrées, ce qui nous permet à chaque instant de pousser ces mises à jour au moteur de règles.

Le moteur de règles. L'évaluation des règles déclenchées par le moniteur adopte une stratégie classique pour les requêtes Datalog récursives [BR86]. Pour chaque événement, les règles produisent des documents à partir des collections extensionnelles, jusqu'à ce qu'aucun nouveau document ne puisse être produit. Notre contexte spécifique (appel de fonction, documents complexes, matérialisation) nécessite quelques adaptations pour la stabilité du système.

Lors de l'évaluation de fonctions externes comme prédicats, celles-ci peuvent produire un, voire plusieurs documents qui seront insérés dans la base de données. La conséquence est la génération de nouveaux événements qui déclenchent eux-mêmes de nouvelles règles. Comme présenté dans la section 4.2.1, pour éviter les cycles, nous vérifions qu'une règle n'est pas déclenchée plus de trois fois par un événement donné. Même si cela restreint le système, cela ne constitue pas une limitation fonctionnelle.

Pour relier les événements aux règles, une table de hachage relie les prédicats associés aux collections mises à jour, et à chaque prédicat correspond une liste de règles dans lesquelles il apparaît. Ainsi, pour chaque événement e de l'*oplog*, nous identifions la collection C impliquée et les règles associées. Pour chaque règle, $C(e)$ est considéré comme base de fait. Les variables associées sont affectées, réduisant l'espace de recherche de la règle, et si tous les faits sont vérifiés, la tête de la règle est vraie et donc instanciée.

5.2.2 Base de données et Bureau virtuel

Afin d'intégrer intuitivement les données dans l'espace de travail de l'utilisateur, nous avons implémenté une interface virtuelle pour le bureau [CdMR+12a] (*Virtual Desktop Interface* - VDI). Celle-ci permet de naviguer dans l'arborescence de répertoires, accéder aux fichiers et leur contenu, effectuer des requêtes sur fichiers, manipuler les métadonnées et manipuler les fichiers d'une collection. Le VDI repose sur la librairie FUSE [Sze17] (*Filesystem in USErspace*) qui nous permet de capturer des appels système.

La librairie FUSE permet d'intégrer l'interface de la base de données sans avoir à modifier le noyau du système d'exploitation. Il capture les requêtes systèmes d'un répertoire monté avec FUSE, tel que : *open*, *read*, *write*... Un tel événement est traduit dans le VDI par une requête sur une collection en utilisant une règle avec le prédicat intentionnel "answer". Le résultat de la requête est transmis au noyau avec les informations demandées par l'interaction de l'utilisateur.

Chaque appel système est accompagné de deux informations que nous pouvons exploiter : le répertoire courant (dénommé par $current), et le chemin complet vers l'objet demandé ($path_name).

Navigation. L'accès au répertoire *$current* (via *readdir()*) génère deux requêtes, une sur Folder pour retrouver les sous-répertoires virtuels de *$current* et la seconde pour les fichiers virtuels. Les deux requêtes ci-dessous illustrent ces deux requêtes en accédant au répertoire *$current* de la vue "*categories*".

answer($fold) :– **Folder**($fold), $fold.parent=$current, $fold.vfs="categories"
answer($file) :– **File**($file), $file.folder=$current.id

Malgré son apparente simplicité, cette dernière règle s'applique à une vue et nécessite d'être substituée par la définition de la vue `File` avec le VFS "*categories*" (exemple 4.2.4). Le corps de la règle doit remplacer la définition de `File($file)` par la définition de la vue comme ci-dessous. Cette traduction est effectuée par la `VDI` en relation avec le moteur de règles.

```
answer("id": [$f.id, $a.id], "name": $a.title, "ext": $a.image.type, "content": $a.image.content, "folder": $f) :–
        Folder($f), Asset2d ($a), $f.id=$a.category.id, $f.id=$current.id
```

QBE. Les interactions de l'utilisateur avec le système de fichier permettent d'effectuer des requêtes par l'exemple (ou QBE). En effet, en copiant une image dans un VFS "*queries*", FUSE va prendre ce fichier comme requête dont le résultat va remplir le répertoire de tous les Asset2D similaires (section 4.2.2).

Pour cela, le contenu du fichier est transmis par FUSE au `VDI` qui va en extraire les descripteurs (couleur, forme et texture [GJ96, Low04]). Ces descriptions sont ensuite intégrées dans une requête `Map/Reduce` afin de calculer la distance `L2` avec les descripteurs des Asset2D qui ont été matérialisés dans la collection "*descriptors*". Nous obtenons la somme des distances normalisées de ces trois vecteurs. Il sera filtré en ne gardant que ceux dont la distance est proche de l'image "requête" et trié par pertinence.

5.3 Interrogation d'un corpora virtuel

Afin de faciliter l'expressivité du langage d'interrogation de partitions, nous avons abstrait la notation musicale (section 3.3). De fait, une requête XQuery ne peut pas être évaluée directement sur le document XML original. Cette évaluation se fait donc sur des instances virtuelles. Traditionnellement, deux choix d'implémentation sont offerts : une traduction de la requête appelée *mapping* sur le modèle original (dite *Global As View* [Ull97, GMUW00]) ou une matérialisation dans le modèle de données abstrait.

5.3.1 *Mapping* de partitions

Pour le *mapping*, nous avons mixé ces deux solutions en créant des vues semi-virtuelles. En effet, la structure d'un opus avec une arborescence permet de séparer les métadonnées de la partition et les voix. Les métadonnées seront matérialisées et un *mapping* des voix sera effectué sur le format original.

Pour traduire le format d'encodage de la partition, nous avons défini un *mapping* en XML Schema (cf. [FSRT16d]) pour chaque corpus permettant de définir la représentation *virtuelle* de chaque opus. Ce schéma facilite l'harmonisation des partitions par rapport aux encodages d'origine et la création de liens avec les voix. Les instances virtuelles sont alors traduites et stockées dans une base de données XML.

La matérialisation de l'opus permet l'interrogation avec un langage comme XQuery. La recherche de partitions en est ainsi facilitée et optimisée grâce à l'indexation des éléments matérialisés. Quant à la virtualisation des voix, elle permet de ne pas dénaturer le format d'origine, d'éviter les problèmes de cohérences, et de laisser la traduction de la notation musicale aux opérateurs dédiés de la `ScoreAlg`.

5.3.2 Intégration d'une collection semi-virtuelle

Le système `Scorelib` intègre l'évaluation de requêtes `ScoreQL` avec la `ScoreAlg`, avec le *mapping* des partitions en vScores. Pour cela, il repose sur la base XML `BaseX`[1] ainsi qu'une librairie Python pour manipulation de la notation musicale : `music21`[2][CA10]. L'avantage de cette architecture est sa flexibilité par l'ajout de fonctions en XQuery. L'architecture présentée dans la figure 5.9 résume les principaux composants de l'évaluation. Les données sont stockées dans deux collections XML : la collection semi-virtuelle (un format par corpus) contenant les opus, et la collection des partitions sérialisées. Chaque instance de la première est liée à une instance de la seconde par les voix.

L'évaluation d'une requête s'effectue de la manière suivante : (1) une requête XQuery, sur la collection matérialisée, retrouve les opus correspondants aux prédicats de la clause `where`, (2) pour chaque opus les voix demandées sont extraites par un *mapping* sous forme d'une requête XQuery dédiée. Ensuite, (3) ces voix sont traduites dans notre modèle de données grâce à l'outil `music21`. Cette traduction permet par la

1. `http://basex.org`
2. `http://web.mit.edu/music21`

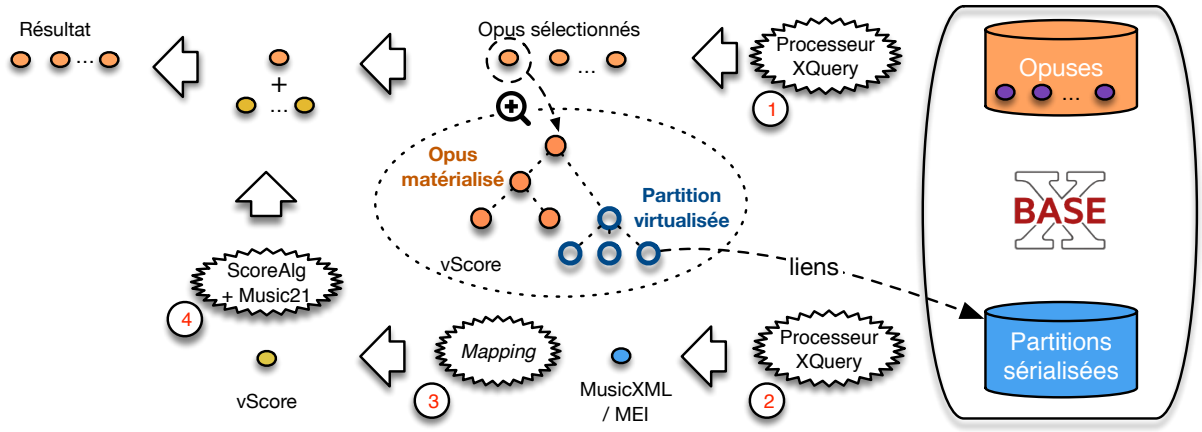


FIGURE 5.9 – Architecture de Scorelib

suite de (4) manipuler les partitions avec les opérateurs de la ScoreAlg implémentés en Java, comme le permet l'architecture de BaseX. music21 sert ici de boîte à outils pour les manipulations élémentaires sur la notation musicale. Nos opérateurs permettent de garantir la cohérence des manipulations.

Il est important de noter que notre système se comporte d'une manière analogue à *ActiveXML* [ABM08] où l'on active le contenu de documents XML à la demande en appelant un service externe à la collection.

Réécriture de requêtes. Afin d'optimiser le traitement des requêtes, une réécriture de requêtes est nécessaire. Pour cela, nous avons proposé des règles d'équivalence [FSRT17] d'expressions algébriques inspirées des règles du modèle relationnel. La table 5.5 montre ces règles qui ont pour contrainte de préserver la structure finale de la partition et les séries d'événements qui la composent.

Ainsi, les règles R1 à R9 s'intéressent donc à une réécriture locale des expressions produites pour améliorer les manipulations de partitions. Les règles R10 et R11 ont la particularité de pousser les projections et les sélections à travers l'opération de *mapping* M vers la requête XQuery. Cette optimisation est indispensable, car les expérimentations effectuées ont montré que l'étape la plus coûteuse restait ce *mapping* de la partition vers music21. Grâce à l'optimisation, nous avons pu limiter le nombre de voix (explicite dans la requête) et la quantité d'événements à traduire (lorsque la sélection est exprimable en XQuery).

L'heuristique résultante de notre processus d'optimisation est composée de deux étapes : 1) pousser les sélections (R1-R3) et les projections (R4-R6) au maximum sur les *vScore*, 2) pousser les sélections sur les partitions sérialisées (R10), la projection des voix étant déjà intégrée à la requête (XPath sur la structure).

Nom	Règle	Commentaire
R1	$\sigma_{F_1}(\sigma_{F_2}(S)) \equiv \sigma_{F_1 \wedge F_2}(S))$	Composition et décomposition de sélections
R2	$\sigma_F(S_1 \oplus S_2) \equiv \sigma_F(S_1) \oplus \sigma_F(S_2)$	Idem
R3	$\sigma_F(S_1 \circ S_2) \equiv \sigma_F(S_1) \circ \sigma_F(S_2)$	Idem
R4	$\mu_{v_i \ldots v_n}(\sigma_F(S)) \equiv \sigma_F(\mu_{v_i \ldots v_n}(S))$	NB : σ est appliqué à tous les événements d'une partition, d'où l'équivalence
R5	$\mu_{v_i \ldots v_n}(S_1 \oplus S_2) \equiv \mu_{v_i \ldots v_j}(S_1) \oplus \mu_{v_{j+1} \ldots v_n}(S_2)$	Avec $v_{i..j} \in S_1, v_{j+1..n} \in S_2$
R6	$\mu_{v_i \ldots v_n}(S_1 \circ S_2) \equiv \mu_{v_i \ldots v_n}(S_1) \circ \mu_{v_i \ldots v_n}(S_2)$	La fusion commute avec la projection
R7	$(S_1 \oplus S_2) \circ S_3 \equiv (S_1 \circ \mu_1(S_3)) \oplus (S_2 \circ \mu_2(S_3))$	Une synchronisation peut être appliquée à des ensembles disjoints de voix
R8	$(S_1 \oplus S_2) \oplus S_3 \equiv S_1 \oplus (S_2 \oplus S_3)$	Associativité de la synchronisation
R9	$(S_1 \circ S_2) \circ S_3 \equiv S_1 \circ (S_2 \circ S_3)$	Associativité de la fusion
R10	$[\mu_{v_i \ldots v_n}(S) \leftarrow M \leftarrow q_S] \equiv [S \leftarrow M \leftarrow \mu_{v_i \ldots v_n}(q_S)]$	La projection des voix peut être appliquée dans la requête XQuery
R11	$[\sigma_F(E) \leftarrow M \leftarrow q_S] \equiv [S \leftarrow M \leftarrow \sigma_F(q_S)]$	Les sélections exprimables en XQuery peuvent être extraites du mapping M.

TABLE 5.5 – Règles de réécriture

5.4 Conclusion

Ce chapitre finalise l'approche abordée dans mes travaux sur l'interrogation de données. Dans cette troisième étape, nous avons pu voir l'implémentation des manipulations présentées dans le chapitre précédent et les choix d'optimisation qui ont été pris pour pouvoir passer à l'échelle. Pour résumer :

— Nous avons vu une approche d'optimisation multi requêtes de souscriptions basée sur la factorisation des opérations communes. La spécificité de l'approche est de proposer un algorithme et une structure de données pour simplifier l'espace de recherche grâce à un arbre de *Steiner*.

— Pour l'optimisation de requêtes larges pour le Pub/Sub, nous avons étudié différentes techniques d'indexation dont le principe est de factoriser les termes en communs. Un modèle analytique et des expérimentations ont pu montrer que l'approche arborescente est volumineuse, tandis que l'utilisation du compteur, bien que coûteuse, reste plus flexible (correspondances partielles).

— La nouveauté et la diversité sont optimisées à la fois en milieu centralisé et distribué, ce qui permet de montrer l'importance du choix du modèle physique et la convergence des traitements distribués.

— La mise en œuvre d'un moteur de règles `Datalog` est réalisée dans un contexte distribué en produisant un moteur de *Trigger* pour `MongoDB`. La matérialisation des collections intentionnelles (hormis *answer*) améliore les performances de requêtes, tout en restant mise à jour grâce au moteur de règles. L'évaluation de requêtes sur le contenu (*answer*) est effectuée à l'aide de requêtes Map/Reduce.

— L'interrogation de la notation musicale avec un modèle de données dédié est intégrée à l'aide d'une semi-matérialisation. La décomposition de cette matérialisation permet un découplage des opérations entre la structure des opus et les opérations sur la notation musicale (séries temporelles) qui peuvent être déléguées à une librairie dédiée.

Dans mon approche, l'optimisation de ces systèmes s'est concentrée essentiellement sur le choix du *modèle physique de données*. En effet, le choix d'intégration de ce modèle logique a un fort impact sur le traitement des données : 1) un modèle de flux pipeliné *query-driven* produisant des graphes que l'on peut factoriser (section 5.1.1), 2) trois modèles de flux indexés *query-driven* pour mixer flexibilité et factorisation (section 5.1.2), 3) deux modèles de flux fenêtrés *item-driven* facilitant la mutualisation des micro traitements, le premier en centralisé (section 5.1.3) et le second en distribué (section 5.1.4), 4) un modèle distribué orienté documents dont les vues (produites par des règles) sont matérialisées à des fins de performances (section 5.2.1), 5) un modèle avec une vue semi-matérialisée combinant structures et séries temporelles, pouvant séparer les opérations sur les séries, des filtres sur la structure (section 5.3.2).

Ainsi, les techniques d'optimisation que j'ai pu utiliser reposent sur quatre familles d'optimisation : l'indexation (accès privilégié), la matérialisation (données précalculées), la factorisation (mutualisation des ressources) et la distribution (parallélisation des calculs). Toutefois, certaines contraintes doivent être prises en compte pour le bon fonctionnement de ces techniques : les mises à jour (index, distribution et matérialisation), de complexité de mise en œuvre (factorisation et distribution) et de choix du modèle physique de données (distribution).

D'autre part, l'expression des manipulations a également un impact sur les possibilités d'optimisation. En effet, les expressions algébriques, grâce à leur forme close, permettent d'effectuer des réécritures. De fait, à l'aide de formules de coût et d'une heuristique, un plan d'exécution peut être optimisé (sections 5.1.1 et 5.3.2).

Toutes ces optimisations reposent sur un modèle de coût permettant d'orienter le choix de l'optimiseur : meilleure couverture de l'arbre de *Steiner*, factorisation des souscriptions de l'index, distribution des items sur un ensemble de serveurs, projeter et filtrer les voix d'une partition.

Ainsi, l'optimisation d'une base de données est un processus à combinaisons multiples pour lequel il est nécessaire de faire le choix d'un modèle physique de données, d'une structure efficace pour accéder aux données et d'un modèle de coût pour orienter l'évaluation de requêtes. Cette combinaison est elle-même soumise à la contrainte due à la définition du modèle logique de données et des possibilités de manipulations offertes par le langage d'interrogation.

Chapitre 6

Conclusion et Perspectives

Dans ce mémoire, nous avons étudié trois différents types de données : des flux d'items avec RSS, des assets graphiques et des partitions de musiques en XML. J'ai donc décomposé chacune des solutions abordées durant mes travaux de recherches en trois étapes fondamentales : le *modèle de données*, le *langage de manipulation* et l'*optimisation*.

À chaque étape, plusieurs choix ont dû être faits, ayant des impacts sur le système et les autres étapes. Cette démarche m'a permis d'appréhender la conception d'une base de données dans son ensemble afin d'effectuer les choix appropriés en fonction des besoins, aussi bien en matière de langage d'interrogation que de contraintes d'intégration et d'optimisation.

6.1 Entre modélisation et optimisation

Pour comprendre cette démarche, il faut revenir à l'architecture ANSI-SPARC [TK78] qui a permis de séparer la modélisation de données, des aspects physiques comme l'intégration ou l'optimisation. Cet aspect fondamental est indispensable lors de la conception d'une base de données relationnelle.

Toutefois, les évolutions des besoins applicatifs, liés à la combinaison hétérogénéité/volume/débit des données, remettent en cause cette séparation et donnent lieu à de nouvelles fonctionnalités pour les bases de données [CAB+13]. En effet, il devient difficile de s'abstraire des spécificités des données tout en ignorant les contraintes physiques. Ainsi, les choix effectués sur le modèle ont des conséquences sur le langage et l'intégration, qui dans ces contextes spécifiques, impliquent des problèmes d'expressivité ou de performances. Il n'est alors plus possible d'ignorer les aspects physiques lors de l'étape de modélisation.

Mes travaux de recherche m'ont permis d'appréhender ce problème de conception en me positionnant entre la modélisation et l'optimisation. En prenant en compte les besoins du système tout aussi bien fonctionnels que techniques, il est possible de proposer une solution qui répondra au mieux au problème. Dans un tel contexte, ce choix doit être un compromis entre la structure du modèle, les manipulations de données et les possibilités d'optimisation. Ma spécificité dans ce domaine est la capacité à positionner le *curseur de conception* sur chacune de ces trois étapes :

— Le modèle de données : choix allant d'un typage fort à des données non structurées.

— Le langage de requêtes : d'un langage déclaratif à un langage naturel.

— L'optimisation avec l'intégration d'un panel de techniques : indexation, distribution, vues matérialisées, factorisation, dénormalisation, compression, etc.

Solutions alternatives. La difficulté de cette approche est de montrer que les bases de données obtenues sont les plus efficaces, ou les plus appropriées. Je propose donc d'étudier des solutions alternatives des projets sur lesquels j'ai travaillé, avec d'autres langages ou techniques d'optimisation et constater qu'elles ne seraient pas appropriées :

— Prenons l'exemple des requêtes structurées sur les flux RSS, il serait possible d'utiliser des langages déclaratifs, tels que `SQL` ou `XQuery`, pour exprimer les manipulations. Pour cela, il faudrait étendre le langage [LSW+04, BFF+07] pour faciliter la manipulation de flux et l'expression de fenêtres sur les items. Toutefois, ces extensions reposent principalement sur le concept de relations "temporelles" vues comme des *snapshots* sur le flux. L'implémentation traditionnelle de ces *snapshots* correspond à une évaluation itérative du flux (tous les X temps ou X items) pour réduire le temps de calcul global. Deux problèmes temporels surviennent : 1) les opérateurs deviennent bloquants (temporairement), réduisant la fluidité des notifications, et 2) chaque fenêtre, étant trop spécifique, empêche la composition/factorisation de celles-ci sur l'ensemble des souscriptions. Notre proposition d'implémentation, les files d'attente pour les opérateurs et les jointures d'annotations, répond tout particulièrement à cette problématique.

D'autre part, ces extensions mélangent dans le langage les concepts de publications et de souscriptions sur les flux. Ainsi, il devient complexe de dissocier la gestion du flux, de la manipulation des items. En conséquence, le nombre de possibilités de composition d'opérateurs est réduit pour une même source. En effet, les flux changent de nature et empêchent toute possibilité d'équivalence lors de réécritures de plans d'exécution, en particulier pour l'étape de factorisation des opérateurs ;

— Proposer des souscriptions avec des requêtes `Datalog` impliquerait un déclenchement des règles en cascade pour chaque nouvel item du flux. Pour pouvoir passer à l'échelle, une factorisation est là aussi nécessaire. [TGL10] propose une structure optimisant la combinaison de prédicats communs entre requêtes, y compris dans le cas de composition de règles. Toutefois, la complexité d'expression des fenêtres et les manipulations de flux rendent cette solution difficilement exploitable.

— Concernant l'optimisation des flux textuels, nous avons proposé un ordre total des termes avec le ROT. L'utilisation d'un arbre de *Steiner* est envisageable pour déterminer la factorisation optimale des termes afin d'en réduire le coût de traitement global. Malheureusement, l'espace de recherche lié au nombre de combinaisons de mots (taille du vocabulaire, section 3.1.1) rend cette solution irréalisable. En effet, cet algorithme NP-complet ne passerait pas à l'échelle (*p. ex.*, [GN12] indexe un petit ensemble d'items et non des souscriptions). De plus, la solution sera difficile à maintenir lors de l'ajout de nouvelles souscriptions.

— Dans le cadre de manipulations d'assets graphiques, les règles `Datalog` nous permettent de définir des contraintes fortes sur des collections et de favoriser la composition de manipulations. L'ensemble des règles déclenchées par un événement produit une chaîne d'exécution que l'on pourrait considérer comme un *workflow* "logique". L'intérêt de cette approche est qu'aucun lien explicite n'est défini dans cette chaîne de traitements, le moteur de règles gère l'intégrité du système, et ce, dans un contexte distribué pour la distribution de calculs.

  Nos partenaires industriels sur le projet `Polymathic` ont étudié l'implémentation plus classique du système sous forme d'un *workflow* "physique". À court terme, quelques chaînes de productions simples ont été produites. Toutefois, au vu du nombre de contraintes de typage (vérification des formats d'entrées de chaque opération), de transformations (extraction de descripteurs d'images, génération des catégories, etc.), la complexité de maintenance de l'ensemble des workflows est devenue trop grande. Le résultat a produit de nombreuses répétitions de traitements (une extraction par source d'image), des délais supplémentaires dans la vérification des formats d'entrées, et des problèmes de performances lors de l'évaluation d'un grand nombre de traitements. La solution ne passait plus à l'échelle (aussi bien humain qu'opérationnel).

— De plus, le calcul des recherches par le contenu (images) aurait pu être implémenté à l'aide de structures d'index multidimensionnels. Pour chaque image, nous avons utilisé les descripteurs de formes, de couleurs et de textures ce qui est déjà traité dans la littérature. Toutefois, deux spécificités du projet `Polymathic` rendent cette solution difficilement intégrable : 1) le nombre de dimensions est voué à grandir en intégrant de nouveaux descripteurs et de nouveaux types d'images (*ex.*, Vidéo, *Motion Capture*), 2) un asset peut être composé de plusieurs médias (un personnage avec son équipement). Cette combinaison impose la création d'index multiples qui sont interrogés indépendamment. Les résultats sont ensuite combinés pour trouver les assets répondant à la requête, tout en appliquant un score de similarité global.

  Notre approche intègre chaque descripteur extrait pour les assets associés et les distribue dans un cluster. De fait, cette solution apporte de la flexibilité dans les descripteurs produits tout en facilitant un passage à l'échelle horizontal.

— Pour les partitions musicales, appliquer `XQuery` sur le format d'origine a déjà été proposé [LSW+04, BBC11] et montre à quel point exprimer des manipulations de la notation demande deux expertises métiers : musicologie et XQuery. De fait, il est nécessaire de créer une librairie de fonctions dédiées à chaque format (MusicXML, MEI) et d'en vérifier la cohérence. Grâce à l'abstraction de notre approche, nous pouvons déléguer ces fonctions à des librairies développées par des experts

(music21), faciliter l'expression de manipulations simples, et nous concentrer sur l'optimisation de ces traitements avec des opérateurs de haut niveau.

— Il aurait également pu être possible d'utiliser des règles à la Datalog pour manipuler la notation musicale. Dedalus [AMC+11] propose une approche similaire impliquant la combinaison de règles. Toutefois, cette solution reste très opérationnelle pour définir des chaînes de transformations sur des séries temporelles. De fait, l'expression d'une requête est bien moins déclarative et trop proche du format sous-jacent. La conséquence est un outil dédié aux experts, qui ne peut se focaliser sur l'optimisation de la séquence de règles produites.

6.2 Recherches actuelles et futures

Mes travaux actuels reflètent cette approche avec trois autres types de données pour lesquels j'oriente ce curseur de conception vers l'expression de motifs sur des graphes de données appliqués à la propagation de calculs (score, similarité, etc.).

6.2.1 Ontologie et qualité de la notation musicale, vers l'interrogation de graphes

En reprenant la problématique de l'exploitation de la notation musicale, nous avons souhaité pouvoir exprimer des concepts musicaux sur lesquels nous pourrions évaluer des mesures de qualité de la partition, en fonction de son contexte. Pour cela, nous avons récemment proposé l'ontologie MusicNote [1] pour la notation musicale [SSCHR+17, SSCGH+17] qui, à l'instar de notre modèle de données (section 3.3), formalise des concepts de notations d'une partition et les liens entre ces concepts. Les éléments d'une partition sont alors modélisés en *faits RDF* [MMM+04] grâce à une procédure XSLT [XSL99].

L'interrogation de cette base de faits s'intéresse à la validation des concepts musicaux. Pour cela, nous avons travaillé avec les musicologues, pour définir les règles métiers dans le cadre de la notation musicale. Ces règles sont contextuelles et donnent lieu à la définition d'extensions de l'ontologie pour chaque concept clé. Nous l'avons appliqué aux dissonances du contrepoint du 16^e siècles. L'ontologie a donc été étendue aux concepts de dissonances, et nous avons exprimé des *règles* en SWRL [HPSB+04] permettant d'effectuer la validation de faits, voire inférer des faits. Ainsi, des faits de dissonances pourront être ajoutés à la base dans le cas où les faits d'une partition ne satisferaient pas les règles du contrepoint.

Au niveau de l'*implémentation*, le principe a été testé sur *Protégé* [2] pour tester les règles de dissonances sur des partitions en RDF. Toutefois, pour approfondir à la fois l'expressivité des règles et l'efficacité de l'approche, je m'intéresse à l'intégration de cette base en les représentant comme graphes de faits musicaux dans neo4j [3] [RWE15]. Ainsi, nous pouvons imaginer des règles comme "motifs musicaux", capables de trouver les instances correspondantes, créer de nouveaux liens et nœuds dans le graphe, définir des calculs de distances entre le motif musical et la partition, et en définitive produire des scores de qualités de partitions, voire de corpus musicaux.

6.2.2 Recommandations pour le *micro blogging*

Je travaille aussi sur la recommandation de messages sur des sites de micro blogging comme Twitter [4]. La thèse de Quentin Grossetti s'attaque au problème de recommandation de messages en temps réel de manière efficace sur un énorme graphe d'utilisateurs. Pour cela, le *modèle de données* repose sur le graphe d'utilisateurs et leur historique de messages, ce qui détermine leur profil. Le modèle proposé est un méta graphe *SimGraph* qui exploite la similarité des historiques entre les utilisateurs, ainsi que leur proximité dans le graphe, cette notion appelée homophilie [GdMT17b]. L'avantage apporté par *SimGraph* est de réduire considérablement le nombre d'arrêtes, tout en les pondérant par un score de similarité.

Aucune requête explicite n'est formulée sur ce modèle. En effet, le profil de l'utilisateur correspond aux données et l'interaction avec un message (*c.-à-d.*, *retweet*) lance une *requête implicite* sur le graphe pour

1. http://cedric.cnam.fr/isid/ontologies/MusicNote.owl
2. https://protege.stanford.edu/
3. https://neo4j.com
4. https://twitter.com

trouver les utilisateurs intéressés. Le score d'un tweet pour un utilisateur donné est déterminé par les similarités combinées aux autres "*retweets*" par propagation dans *SimGraph*. Ainsi, un utilisateur obtient une liste de messages triés par score de recommandation venant de son entourage.

Le graphe est *implémenté* dans un système centralisé avec 2,2M de nœuds [GdMT17a, GCdMT18]. *SimGraph* réduit le temps de calcul grâce à sa propriété exploitant l'homophilie, toutefois, le temps de convergence du score par propagation dans le graphe peut être long. Afin de faciliter cette convergence, un seuil dynamique est établi sur la popularité des messages (faibles et fortes). D'autre part, le taux de rafraîchissement des scores (fenêtre de temps pour les calculs) a aussi un impact sur le temps de calcul. Nous montrons que *SimGraph* obtient des temps de réponse analogues à *Twitter* [SJB+16], mais avec des messages plus orientés sur les comportements similaires que la popularité.

Comme perspective, l'exploitation du graphe dans son ensemble peut être bénéfique sur un aspect complémentaire à l'homophilie pour la recommandation. Les communautés (*c.-à-d.*, sous-graphes) ont pour effet de cloisonner l'information. Ce comportement est plus connu sous le nom de "bulles filtrantes" [NHH+14] avec les effets "*Trump*" ou "*Brexit*". Il est possible de détecter ces bulles d'information en créant un méta graphe intercommunautaire qui pourra servir pour casser le cloisonnement et propager l'information. L'idée serait donc d'exploiter la topologie du graphe (*p. ex.*, *Louvain* ou *InfoMap* [HKK16]), ainsi que la sémantique des messages des utilisateurs dans de telles communautés, et de les combiner pour définir un méta graphe de "bulles" reliées par des liens de similarités produit à la fois par les connexions réelles et les corrélations sémantiques. L'expression de motifs de propagation dans ce méta graphe permettrait alors à la fois de détecter des cloisonnements, mais aussi de proposer un facteur complémentaire à notre système de recommandation.

6.2.3 Interrogation de bases prosopographiques

Le projet ANR QalHis'17 s'intéresse à la constitution d'une base de données prosopographique, dans laquelle est répertoriée un ensemble de fiches historiques sur des flux migratoires en Europe au moyen-âge. Nous cherchons dans ce projet à rapprocher des fiches constituées par différents historiens en Europe afin de faire évoluer les méthodologies de recherche et de faciliter l'exploitation des connaissances de ce domaine. Le principal problème vient du fait que les données ne sont pas fiables car incomplètes, le nom d'une personne change en fonction de l'endroit où elle se trouvait, leurs titres varient d'un pays à l'autre, dépend de l'historien qui a produit la fiche, etc. De fait, être capable de faire le rapprochement de deux fiches pour reconstituer le parcours d'une personne revient à faire un long travail d'étude pour un historien.

Mon approche dans ce contexte est de décomposer les fiches avec différents concepts de connaissances, produisant alors des faits que l'on peut représenter sous forme de graphe. Le résultat sera alors un graphe très peu connexe, car peu d'informations sont renseignées et liées entre elles, si ce n'est les lieux et les titres, éventuellement les "maîtres". Toutefois, il est envisageable de créer des liens avec un degré de confiance après certaines déductions sur les données et densifier le graphe. Cette enrichissement peut être complété par l'intégration de diverses sources d'information de bases historiques annotées, de les interroger et de trier les fiches candidates par ordre de pertinence [MLVP16].

Pour déduire des liens entre les données, j'envisage de produire un langage de haut niveau à base de requêtes par similarité de motifs "prosopographiques" à l'aide de scores de pertinence. Cela permettra de produire des sous-graphes de faits candidats pour un "rapprochement de fiches". Cette étape pourra ainsi constituer une nouvelle méthode de recherche dans une base prosopographique tout en gardant un degré de flexibilité dans la façon d'exprimer la recherche. La difficulté se porte donc dans la définition de similarités de motifs avec l'intégration de traductions et de contextes spécifiques.

Chapitre

Bibliographie

[AAC+08] S. Abiteboul, T. Allard, P. Chatalic, G. Gardarin, A. Ghitescu, F. Goasdoué, I. Manolescu, B. Nguyen, M. Ouazara, A. Somani, N. Travers, and G. Vasile. WebContent : Efficient P2P Warehousing of Web Data. In *VLDB*, pages 1428–1431, 2008.

[AAYIM13] S. Abbar, S. Amer-Yahia, P. Indyk, and S. Mahabadi. Real-time recommendation of diverse related articles. In *WWW*, pages 1–12, 2013.

[ABGA11] S. Abiteboul, M. Bienvenu, A. Galland, and É. Antoine. A Rule-Based Language for Web Data Management. In *PODS*, pages 293–304, 2011.

[ABM08] S. Abiteboul, O. Benjelloun, and T. Milo. The active XML project : an overview. *VLDB*, 17(5) :1019–1040, 2008.

[ABW06] A. Arasu, S. Babu, and J. Widom. The CQL continuous Query Language : Semantic Foundations and Query Execution. In *VLDB*, pages 121–142, 2006.

[ACTV10] B. Amann, J. Creus, N. Travers, and D. Vodislav. ROSES et l'agrégation Web avancée. In *BDA*, volume 26, pages 1–6, 2010.

[AG91] S. Abiteboul and S. Grumbach. A rule-based language with functions and sets. *TDS*, 16(1) :1–30, 1991.

[AH88] S. Abiteboul and R. Hull. Data Functions, Datalog and Negation. In *SIGMOD*, pages 143–153, 1988.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AHW95] A. Aiken, J. M. Hellerstein, and J. Widom. Static analysis techniques for predicting the behavior of active database rules. *TDS*, 20(1) :3–41, 1995.

[AK11] A. Angel and N. Koudas. Efficient diversity-aware search. In *Proceedings of the ACM International Conference on Management of data (SIGMOD'11)*, pages 781–792. ACM, 2011.

[Ama17] Amazon S3fs. `https://aws.amazon.com/fr/s3/`, 2017. Accédé le 2017-11-11.

[AMC+11] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. Sears. Dedalus : datalog in time and space. In *Datalog'10*, pages 262–281, 2011.

[AMM08] S. Ames, C. Maltzahn, and E. L. Miller. QUASAR : Interaction with File Systems Using a Query and Naming Language. Technical report, Univ. of California, Santa Cruz, 2008.

[Ato07] Atom. Atom : The Atom Publishing Protocol, 2007. J. Gregorio, ed. and Google and B. de Hora, ed. and NewBay Software. `https://tools.ietf.org/html/rfc5023`.

[AV91] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *JCSS*, 43(1) :62–124, 1991.

[Bal96] M. Balaban. The music structures approach to knowledge representation for music processing. *Computer Music Journal*, 20(2) :96–111, 1996.

[Ban12] K. Banker. *MongoDB in Action*. Manning Publications Co., 2012.

[BBB+10] V. Bavi, T. Beirne, N. Bone, J. Mohr, and B. Neal. Comparison of Document Similarity Metrics, 2010. Computer Science Dept., Western Washington University Information Retrieval.

[BBC11] M.-A. Baazizi, N. Bidoit, and D. Colazzo. Efficient encoding of temporal xml documents. In *TIME*, pages 15–22. IEEE Computer Society, 2011.

[BCP98] E. Baralis, S. Ceri, and S. Paraboschi. Compile-Time and Runtime Analysis of Active Behaviors. *IEEE Trans. Knowl. Data Eng.*, 10(3) :353–370, 1998.

[BD91] P.J. Brockwell and R.A. Davis. *Time Series : Theory and Methods*. Springer Series in Statistics. Springer, 1991.

[BDR04] J. Bailey, G. Dong, and K. Ramamohanarao. On the decidability of the termination problem of active database systems. *Theor. Comput. Sci.*, 311(1-3) :389–437, 2004.

[BFF+07] I. Botan, P.M. Fischer, D. Florescu, D. Kossman, T. Kraska, and R. Tamosevicius. Extending XQuery with Window Functions. In *VLDB*, pages 75–86, 2007.

[BJC+04] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. A. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *SIGIR*, pages 321–328, 2004.

[BMPT09] S. Brandt, C. Maltzahn, N. Polyzotis, and W.-C. Tan. Fusing Data Management Services with File Systems. In *PDSW*, pages 42–46, 2009.

[BR86] F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *SIGMOD*, pages 16–52, 1986.

[BS74] A. Bookstein and D.R. Swanson. Probabilistic Models for Automatic Indexing. *Journal of the American Society for Information Science*, 25(5) :312–318, 1974.

[BYRN99] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[CA10] M. S. Cuthbert and C. Ariza. Music21 : A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In *ISMIR*, pages 637–642, 2010.

[CAB+13] C. Collet, B. Amann, N. Bidoit, M. Boughanem, M. Bouzeghoub, A. Doucet, D. Gross-Amblard, J.-M. Petit, M.-S. Hacid, and G. Vargas-Solar. De la gestion de bases de données à la gestion de grands espaces de données. *ISI*, 18(4) :11–31, 2013.

[CAC+12] J. Creus, B. Amann, V. Christophides, N. Travers, and D. Vodislav. Optimisation de grandes collections de requêtes d'agrégation RSS. *ISI*, pages 1–28, 2012.

[CATV10] J. Creus, B. Amann, N. Travers, and D. Vodislav. Un agregateur de flux rss avancé. In *BDA*, 2010.

[CATV11a] J. Creus, B. Amann, N. Travers, and D. Vodislav. Optimizing large collections of continuous content-based RSS aggregation queries. In *BDA*, pages 1–19, 2011.

[CATV11b] J. Creus, B. Amann, N. Travers, and D. Vodislav. RoSeS : A continuous content-based query engine for RSS feeds. In *DEXA*, volume 6861 of *LNCS*, pages 203–218, 2011.

[CATV11c] J. Creus, B. Amann, N. Travers, and D. Vodislav. RoSeS : A continuous query processor for large-scale RSS filtering and agregation. In *CIKM*, pages 2549–2552, 2011.

[CCC+98] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. In *SODA*, pages 192–200. Society for Ind. and Applied Math., 1998.

[CCD+03] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ : Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003.

[CdMR+12a] C. Constantin, C. du Mouza, P. Rigaux, V. Thion, and N. Travers. A Desktop Interface over Distributed Document Repositories. In *EDBT*, pages 104–107, 2012.

[CdMR+12b] C. Constantin, C. du Mouza, P. Rigaux, V. Thion, and N. Travers. Browse Your Content-Based Distributed Repository ! In *BDA*, pages 1–5, 2012.

[CDTW00] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ : A Scalable Continuous Query System for Internet Databases. In *SIGMOD*, pages 379–390, 2000.

[CGT89] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *TKDE*, 1(1) :146–166, 1989.

[CKC+08] C. L.A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, pages 659–666. ACM, 2008.

[CKSV08] M. Cammert, J. Kramer, B. Seeger, and S. Vaupel. A cost-based approach to adaptive resource management in data stream systems. In *TKDE*, volume 20, pages 230–245, 2008.

[CO90] F. Can and E. A. Ozkarahan. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *TDS*, 15(4) :483–517, 1990.

[Cod90] E. F. Codd. *The Relational Model for Database Management, Version 2*. Addison-Wesley, 1990.

[CPY07] B. Chandramouli, J. M. Phillips, and J. Yang. Value-based notification conditions in large-scale publish/subscribe systems. In *VLDB*, pages 878–889, 2007.

[CRYT10] D. Carmel, H. Roitman, and E. Yom-Tov. On the Relationship Between Novelty and Popularity of User-generated Content. In *CIKM*, pages 1509–1512, 2010.

[DGH+06] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. M. White. Towards expressive publish/subscribe systems. In *EDBT*, pages 627–644, 2006.

[DHI12] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[dMGFS11] O. de Moor, G. Gottlob, T. Furche, and A. J. Sellers, editors. *Datalog Reloaded*, volume 6702 of *LNCS*. Springer, 2011.

[dMT18] C. du Mouza and N. Travers. *Trends and Challenges related to NoSQL data models, O. Pivert*, chapter Relevant Filtering in a Distributed Content-based Publish/Subscribe System, pages 193–226. ISTE, 2018.

[DNT07] T.-T. Dang-Ngoc and N. Travers. Tree Graph Views for a Distributed Pervasive Environment. In *NBiS*, pages 406–415, 2007.

[DP09] M. Drosou and E. Pitoura. Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4) :49–56, 2009.

[DP12a] M. Drosou and E. Pitoura. Disc diversity : result diversification based on dissimilarity and coverage. *VLDB*, 6(1) :13–24, 2012.

[DP12b] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *EDBT*, pages 216–227. ACM, 2012.

[Dro17] DropBox. `https://www.dropbox.com/`, 2017. Accédé le 2017-11-11.

[DSP09] M. Drosou, K. Stefanidis, and E. Pitoura. Preference-aware publish/subscribe delivery with diversity. In *DEBS*, pages 1–12. ACM, 2009.

[FBF13] E. Fares, J.-P. Bodeveix, and M. Filali. Event Algebra for Transition Systems Composition Application to Timed Automata. In *TIME*, pages 125–132. CPS, 2013.

[FJL+01] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering Algorithms and Implementation for Very Fast Pub/Sub. In *SIGMOD*, pages 115–126, 2001.

[FLOB13] D. Fober, S. Letz, Y. Orlarey, and F. Bevilacqua. Programming Interactive Music Scores with INScore. In *Sound and Music Computing*, pages 185–190, Stockholm, Sweden, July 2013.

[FOL12] D. Fober, Y. Orlarey, and S. Letz. Scores level composition based on the guido music notation. In ICMA, editor, *Proc. Int. Computer Music Conference*, pages 383–386, 2012.

[FSRT16a] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. A Digital Score Library Based on MEI. In *(MEC'16) Music Encoding Conference*, pages 1–4, 2016.

[FSRT16b] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Is There a Data Model in Music Notation ? In *TENOR*, pages 85–91. Anglia Ruskin University, 2016.

[FSRT16c] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Querying Music Notation. In *TIME*, pages 1–9, Kongens Lyngby, Denmark, October 2016.

[FSRT16d] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Querying XML Score Databases : XQuery is not Enough ! In *ISMIR*, pages 723–729, 2016.

[FSRT16e] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Vers un Traitement Algébrique de la Notation Musicale. In *JIM*, volume 23, pages 61–70, 2016.

[FSRT17] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Modeling Music as Synchronized Time Series : Application to Music Score Collections. *(IS'18) Information Systems*, pages 1–36, 2017.

[GCdMT18] Q. Grossetti, C. Constantin, C. du Mouza, and N. Travers. An Homophily-based Approach for Fast Post Recommendation in Microblogging Systems. In *EDBT*, pages 1–12, 2018.

[GDH04] E. Gabrilovich, S. Dumais, and E. Horvitz. Newsjunkie : Providing Personalized Newsfeeds via Analysis of Information Novelty. In *WWW*, pages 482–490, 2004.

[GdMT17a] Q. Grossetti, C. du Mouza, and N. Travers. Enhance micro-blogging recommendations of posts with an homophily-based graph. In *BDA'17*, pages 1–10, 2017.

[GdMT17b] Q. Grossetti, C. du Mouza, and N. Travers. Tweet, Retweet et Follower : que recommander et à qui ? In *Workshop AISR*, pages 1–6, 2017.

[GJ96] G. Gimel'farb and A. Jain. On retrieving textured images from an image database. *Pattern Recognition*, 29(9) :1461–?1483, 1996.

[GJSO91] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, Jr. Semantic file systems. In *SOSP*, pages 16–25. ACM, 1991.

[GMUW00] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database System Implementation*. Prentice Hall, 2000.

[GN12] A. Gubichev and T. Neumann. Fast approximation of steiner trees in large graphs. In *CIKM*, pages 1497–1501. ACM, 2012.

[GÖ03] L. Golab and M. T. Özsu. Issues in Data Stream Management. *SIGMOD*, 32(2) :5–14, 2003.

[Goo01] M. Good. *The Virtual Score*, chapter MusicXML for Notation and Analysis, pages 113–124. W. B. Hewlett and E. Selfridge-Field, MIT Press, 2001.

[Goo14] Google Drive. `https://drive.google.com`, 2014. Accédé le 2017-11-11.

[GS03] A. Kumar Gupta and D. Suciu. Stream Processing of XPath Queries with Predicates. In *SIGMOD*, pages 419–430, 2003.

[GSD08] J. Ganseman, P. Scheunders, and W. D'haes. Using XQuery on MusicXML Databases for Musicological Analysis. In *ISMIR*, 2008.

[HDG+07] M. Hong, A. J. Demers, J. Gehrke, C. Koch, M. Riedewald, and W. White. Massively Multi-Query Join Processing in Publish/Subscribe Systems. In *SIGMOD*, pages 761–772, 2007.

[HdMT13] Z. Hmedeh, C. du Mouza, and N. Travers. An Intelligent PubSub Filtering System. In *BDA*, pages 1–5, 2013.

[HdMT15a] Z. Hmedeh, C. du Mouza, and N. Travers. A Real-time Filtering by Novelty and Diversity for Publish/Subscribe Systems. In *SSDBM*, pages 1–4, 2015.

[HdMT15b] Z. Hmedeh, C. du Mouza, and N. Travers. TDV-based Filter for Novelty and Diversity in a Real-time Pub/Sub System. In *IDEAS*, volume 19, pages 136–145, 2015.

[Hel10] J. M. Hellerstein. The declarative imperative : experiences and conjectures in distributed logic. *SIGMOD*, 39(1) :5–19, 2010.

[HKC+12a] Z. Hmedeh, H. Kourdounakis, V. Christophides, C. du Mouza, M. Scholl, and N. Travers. Indexes Analysis for Matching Subscriptions in RSS feeds. In *BDA*, pages 1–20, 2012.

[HKC+12b] Z. Hmedeh, H. Kourdounakis, V. Christophides, C. du Mouza, M. Scholl, and N. Travers. Subscription Indexes for Web Syndication Systems . In *EDBT*, pages 311–322, 2012.

[HKC+13] Z. Hmedeh, H. Kourdounakis, V. Christophides, C. du Mouza, M. Scholl, and N. Travers. Techniques d'indexation de souscriptions pour la syndication web. *ISI*, 18(4) :33–58, 2013.

[HKC+16] Z. Hmedeh, H. Kourdounakis, V. Christophides, C. du Mouza, M. Scholl, and N. Travers. Content-Based Publish/Subscribe System for Web Syndication. *JCST*, 31(2) :357–378, 2016.

[HKK16] P. Held, B. Krause, and R. Kruse. Dynamic clustering in social networks using louvain and infomap method. In *ENIC*, pages 61–68, 2016.

[HPSB+04] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosofand, and M. Dean. SWRL : A semantic web rule language combining OWL and RuleML. W3C Member Submission, 2004.

[HTV+11] Z. Hmedeh, N. Travers, N. Vouzoukidou, V. Christophides, C. du Mouza, and M. Scholl. Everything you would like to know about RSS feeds and you are afraid to ask. In *BDA*, pages 1–20, 2011.

[Hud15] P. Hudak. *The Haskell School of Music – From Signals to Symphonies*. (Version 2.6), 2015.

[Hur02] D. Huron. Music information processing using the humdrum toolkit : Concepts, examples, and lessons. *Computer Music Journal*, 26(2) :11–26, July 2002.

[HVT+11] Z. Hmedeh, N. Vouzoukidou, N. Travers, V. Christophides, C. du Mouza, and M. Scholl. Characterizing Web Syndication Behavior and Content. In *WISE*, pages 29–42. Springer Heidelberg, 2011.

[JA06] S. Jun and M. Ahamad. FeedEx : Collaborative Exchange of News Feeds. In *WWW*, pages 113–122, 2006.

[JBDC+13] D. Janin, F. Berthaut, M. Desainte-Catherine, Y. Orlarey, and S. Salvati. The T-Calculus : towards a structured programing of (musical) time and space. In *Proc. ACM SIGPLAN workshop on Functional art, music, modeling and design (FARM'13)*, pages 23–34, 2013.

[KCC12] M. Keikha, F. Crestani, and W. B. Croft. Diversity in Blog Feed Retrieval. In *CIKM*, pages 525–534, 2012.

[KCM09] A. C. König, K. W. Church, and M. Markov. A Data Structure for Sponsored Search. In *ICDE*, pages 90–101, 2009.

[Knu73] D. E. Knuth. *The Art of Computer Programming, Volume III : Sorting and Searching*. Addison-Wesley, 1973.

[KSSS04] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. FluXQuery : An Optimizing XQuery Processor for Streaming XML Data. In *VLDB*, 2004.

[LC06] R. Levering and M. Cutler. The portrait of a common html web page. In *ACM Symposium on Document Engineering*, pages 198–204, 2006.

[LLM98] Ge. Lausen, B. Ludäscher, and W. May. On Active Deductive Databases : The Statelog Approach. In *Intl. Seminar on Logic Databases and the Meaning of Change, Transactions and Change in Logic Databases*, ILPS, pages 69–106, 1998.

[LMT+05] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. Semantics and Evaluation Techniques for Window Aggregates in Data Streams. In *SIGMOD*, pages 311–322, 2005.

[Low04] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2) :91–110, 2004.

[LS03] A. Lerner and D. Shasha. AQuery : Query Language for Ordered Data, Optimization Techniques, and Experiments. In *VLDB*, pages 345–356, 2003.

[LSW+04] A. Lerner, D. Shasha, Z. Wang, X. Zhao, and Y. Zhu. Fast Algorithms for Time Series with applications to Finance, Physics, Music, Biology, and other Suspects. In *SIGMOD*, pages 965–968, 2004.

[LTWZ05] C. Luo, H. Thakkar, H. Wang, and C. Zaniolo. A Native Extension of SQL for Mining Data Streams. In *SIGMOD*, pages 873–875, 2005.

[LYD+07] X. Li, J. Yan, Z. Deng, L. Ji, W. Fan, B. Zhang, and Z. Chen. A Novel Clustering-Based RSS Aggregator. In *WWW*, pages 1309–1310, 2007.

[MEI17] Music Encoding Initiative. `http://music-encoding.org`, 2017. Accédé le 2017-11-11.

[Mil95] G. A. Miller. WordNet : A Lexical Database for English. *ACM J.*, 38(11) :39–41, 1995.

[MK07] H. H. Malik and J. R. Kender. Optimizing Frequency Queries for Data Mining Applications. In *ICDM*, pages 595–600, 2007.

[MLVP16] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries : a new way of searching. *VLDB*, 25(6) :741–765, 2016.

[MMM+04] F. Manola, E. Miller, B. McBride, et al. Rdf primer. *W3C recommendation*, 10(1-107) :6, 2004.

[MRS08] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. isbn 0521865719.

[MSN11] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets : a streaming-based approach. In *SIGIR*, pages 585–594. ACM, 2011.

[MTX+09] M. Mesiti, M. Traian, Li Xiong, S. Muller, H. Naacke, B. Novikov, G. Raschia, I. Sanz, P. Sens, D. Shaporenkov, and N. Travers. *Proc. EDBT/ICDT Workshops*. ACM, 2009.

[MZ07] S. Ma and Q. Zhang. A Study on Content and Management Style of Corporate Blogs. In *OCSC*, volume 15, pages 116–123, 2007.

[MZV07] T. Milo, T. Zur, and E. Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. In *SIGMOD*, pages 749–760, 2007.

[NHH+14] T.T. Nguyen, P.-M. Hui, F. M. Harper, L. Terveen, and J.A. Konstan. Exploring the filter bubble : The effect of using recommender systems on content diversity. In *WWW*, pages 677–686, 2014.

[NT89] S. A. Naqvi and S. Tsur. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, 1989.

[One17] Microsoft OneDrive For Business. `https://onedrive.live.com/about/business/`, 2017. Accédé le 2017-11-11.

[PC03] F. Peng and S.S. Chawathe. XPath Queries on Streaming Data. In *SIGMOD*, pages 431–442, 2003.

[PDSAT12] D. Panigrahi, A. Das Sarma, G. Aggarwal, and A. Tomkins. Online selection of diverse results. In *WSDM*, pages 263–272. ACM, 2012.

[PFL+00] J. Pereira, F. Fabret, F. Llirbat, R. Preotiuc-Pietro, K. A. Ross, and D. Shasha. Publish/Subscribe on the Web at Extreme Speed. In *VLDB*, pages 627–630, 2000.

[PR03] Y. Padioleau and O. Ridoux. A logic file system. In *USENIX*, pages 99–112, 2003.

[Pre09] Oxford University Press. RT this : OUP Dictionary Team monitors Twitterer's tweets, 2009. `https://blog.oup.com/2009/06/oxford-twitter/`.

[PS06] K. Patroumpas and T. Sellis. *ICSNW*, chapter Window Specification over Data Streams, pages 445–464. Springer Berlin Heidelberg, 2006.

[PvA08] K. Pripužić, I. Podnar Žarko, and K. Aberer. Top-k/w publish/subscribe : finding k most relevant publications in sliding time window w. In *DEBS*, pages 127–138. ACM, 2008.

[QH13] D. Quick and P. Hudak. Grammar-based automated music composition in haskell. In *ACM SIGPLAN workshop FARM*, pages 59–70, 2013.

[RMP+07] I. Rose, R. Murty, P. R. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra : Content-based filtering and aggregation of blogs and rss feeds. In *NSDI*, 2007.

[Rol02] P. Rolland. The Music Encoding Initiative (MEI). In *Proc. Intl. Conf. on Musical Applications Using XML*, pages 55–59, 2002.

[RSS03] RSS Advisory Board. RSS 2.0 : Really Simple Syndication, 2003. Berkman Center for Internet and Society at Harvard Law School. `http://www.rssboard.org/rss-specification`.

[RSV01] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases*. Morgan Kaufmann, 2001.

[RWE15] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases : New Opportunities for Connected Data 2nd Edition*. O'Reilly Media, Inc., 2015.

[Sel88] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13 :23–52, 1988.

[SGFJ13] A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski. Top-k publish-subscribe for social annotation of news. *VLDB*, 6(6) :385–396, 2013.

[SJB+16] A. Sharma, J. Jiang, P. Bommannavar, B. Larson, and J. Lin. GraphJet : Real-time Content Recommendations at Twitter. *VLDB*, 9(13) :1281–1292, 2016.

[SM01] B. Smyth and P. McClave. Similarity vs. Diversity. In *Proc. Int. Conf. on Case-Based Reasoning*, pages 347–361, 2001.

[SMK05] N. Schmidt-Manz and M. Koch. Patterns in search queries. In *Data Analysis and Decision Support*, pages 122–129. Springer Berlin Heidelberg, 2005.

[SN03] G. Steinke and C. Nickolette. Business rules as the basis of an organization's information systems. *Industrial Management & Data Systems*, 103(1) :52–63, 2003.

[SSCGH+17] S. Si-Said Cherfi, C. Guillotel, F. Hamdi, P. Rigaux, and N. Travers. Ontology-Based Annotation of Music Scores. In *K-CAP'17*, pages 1–4, Austin, Texas, USA, 2017.

[SSCHR+17] S. Si-Said Cherfi, H. Hamdi, P. Rigaux, V. Thion, and N. Travers. Formalizing Quality Rules on Music Notation – an Ontology-based Approach. In *TENOR*, pages 1–7, 2017.

[SV89] Y. C. Sagiv and M.Tra Y. Vardi. Safety of datalog queries over infinite databases. In *PODS*, pages 160–171, 1989.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *ACM J.*, 18(11) :613–620, 1975.

[Sze17] M. Szeredi. Filesystem in USErspace. `http://fuse.sourceforge.net/`, 2017. Accédé le 2017-11-11.

[TDN07a] N. Travers and T.-T. Dang-Ngoc. An extensible rule transformation model for XQuery optimization - rules pattern for XQuery tree graph view. In *ICEIS*, pages 351–358, 2007.

[TDN07b] N. Travers and T.-T. Dang-Ngoc. XLive : Integrating Source With XQuery. In *WebIST*, pages 230–233, 2007.

[TDNL07a] N. Travers, T.-T. Dang-Ngoc, and T. Liu. TGV : A Tree Graph View for Modeling Untyped XQuery. In *DASFAA*, pages 1001–1006, 2007.

[TDNL07b] N. Travers, T.-T. Dang-Ngoc, and Tianxiao Liu. Untyped XQuery Canonization. In *APWeb/-WAIM Workshops*, pages 358–371, 2007.

[TGL10] K. Tuncay Tekle, Michael Gorbovitski, and Yanhong A. Liu. Graph queries through datalog optimizations. In *International ACM SIGPLAN PPDP*, pages 25–34. ACM, 2010.

[THV+14] N. Travers, Z. Hmedeh, N. Vouzoukidou, C. du Mouza, V. Christophides, and M. Scholl. RSS feeds behavior analysis, structure and vocabulary. *IJWIS*, 10(3) :291–320, 2014. Top 3 of IJWIS journal papers for 2014.

[TK78] D. Tsichritzis and A. Klug. The ANSI/X3/SPARC DBMS framework report of the study group on database management systems. *Information Systems*, 3(3) :173–191, 1978.

[Tra06] N. Travers. *Optimisation extensible dans un médiateur de données semi-structurées*. PhD thesis, Université de Versailles-St Quentin en Yvelines, 2006.

[Tra12] N. Travers. *Web Data Management*, chapter Putting into Practice : Full-Text Indexing with LUCENE, pages 355–363. Cambridge University Press, 2012.

[Tra17] N. Travers. "Maîtrisez les bases de données NoSQL", OpenClassrooms, 2017. Cours sur le NoSQL sur Openclassrooms.

[TWV05] R. Typke, F. Wiering, and R. C. Veltkamp. A Survey Of Music Information Retrieval Systems. In *ISMIR*, 2005.

[Ull97] J. D. Ullman. *Information integration using logical views*, pages 19–40. Springer Berlin Heidelberg, 1997.

[Vou10] N. Vouzoukidou. On the statistical properties of web search queries. Technical report, ISL, ICS-FORTH, Greece, 2010.

[W3C98] W3C. Extensible Markup Language (XML). World Wide Web Consortium, 1998. `https://www.w3.org/XML`.

[WC96] J. Widom and S. Ceri. Introduction to Active Database Systems. In *Active Database Systems - Triggers and Rules for Advanced Database Processing*, pages 2–41. Springer, 1996.

[WDR06] E. Wu, Y. Diao, and S. Rizvi. High-Performance Complex Event Processing over Streams. In *SIGMOD*, pages 407–418, 2006.

[WGMB+09] S. E. Whang, H. Garcia-Molina, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, and R. Yerneni. Indexing boolean expressions. *VLDB*, 2 :37–48, 2009.

[Whe11] P. Wheatland. Thoth music learning software, v2.5, Accédé le 2017-11-11. `http://scarterfrogs.phpwebhosting.com/thoth.html`.

[Wil85] P. Willett. An algorithm for the calculation of exact term discrimination values. *Inf. Process. Manage.*, 21(3) :225–232, 1985.

[WW10] S. Wang and H. Wang. Business rule management for enterprise information systems. *Information Resources Management Journal (IRMJ)*, 23(1) :53–73, 2010.

[WZ05] H. E. Williams and J. Zobel. Searchable words on the Web. *JODL*, 5(2) :99–105, 2005.

[XQu07] XQuery 3.0 : An XML Query Language. World Wide Web Consortium, 2007. `https://www.w3.org/TR/xquery-30/`.

[XSL99] The Extensible Stylesheet Language Family (XSL). World Wide Web Consortium, 1999. https ://www.w3.org/Style/XSL.

[Yah07] Yahoo! The yahoo! pipes feed aggregator. `https://www.pipes.digital/`, 2007. Clotûre en octobre 2015 - Nouvelle version sur "Re-Live Yahoo Pies", accédé le 2017-11-11.

[Yah08] Yahoo! Yahoo! query language. `https://developer.yahoo.com/yql`, 2008. Accédé le 2017-11-11.

[YGM94] T. W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *TODS*, 19(2) :332–364, 1994.

[YGM99] T. W. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. *TODS*, 24(4) :529–565, 1999.

[YLAY09] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world : diversification in recommender systems. In *EDBT*, pages 368–378. ACM, 2009.

[ZCM02] Y. Zhang, J. Callan, and T. Minka. Novelty and redundancy detection in adaptive filtering. In *SIGIR*, pages 81–88. ACM, 2002.

[ZMTL01] J. Y. Zien, J. Meyer, J. A. Tomlin, and J. Liu. Web Query Characteristics and their Implications on Search Engines. In *WWW*, pages 1–2, 2001.

Abstract

This Habilitation Thesis outlines main results obtained during my research activities carried out as Associate Professor at *Conservatoire National des Arts et Métiers* (CNAM) since 2006. During this period my research interests were driven by querying various types of data : i) textual data flows, ii) graphical assets, and iii) musical scores. I focus especially on the design of data models, query languages and query optimization techniques. Each step was motivated by the way to efficiently enhance querying capabilities for specific data contexts called Domain Specific Languages (DSL). This thesis reports and compares motivations, techniques and results obtained along these lines of research, and proposes an approach of database conception positioned between modeling and query optimization.

Résumé

Cette thèse d'Habilitation à Diriger les Recherches présente les résultats obtenus durant mes activités de recherche en tant que Maître de Conférences au *Conservatoire National des Arts et Métiers* (CNAM) depuis 2006. Durant cette période, mes travaux de recherches se sont dirigés sur la manipulation de différents types de données : i) des flots de données textuelles, ii) des artefacts graphiques, et iii) des partitions musicales. Je me suis particulièrement intéressé à la conception et l'intégration de modèles de données, de langage d'interrogation et de techniques d'optimisation de requêtes. Chaque étape est motivée par la façon d'améliorer l'interrogation des données de manière efficace dans des contextes spécifiques de données (DSL). Cette thèse résume et compare les différentes motivations, techniques et résultats obtenus durant ces années de recherche, pour proposer une approche de conception de bases de données se positionnant entre modélisation et optimisation de requêtes.