Représentation de schémas de bases de données hétérogènes sous forme de métaschémas XML

Amar Zerdazi

Université de Marne-la-Vallée
Institut Francilien d'Ingénierie des Services (IFIS)
Ecole Doctorale Information, Communication, Modélisation et Simulation (ICMS)

D.E.A Information Scientifique et Technique

Option : Méthodes informatiques pour l'organisation et la recherche d'information

Mémoire de DEA

Représentation de schémas de bases de données hétérogènes sous forme de métaschémas XML

(Thème : Intégration de bases de données hétérogènes via XML)

Réalisé par
Amar ZERDAZI

Directrice de Recherche
Myriam LAMOLLE

2002/2003

Laboratoire d'Informatique et de Communication (LINC)
IUT de Montreuil Paris 8
140, Rue de la Nouvelle France
93100 Montreuil

Résumé
Les systèmes de gestion des bases de données ont des structures et des formats de stockages différents pour représenter une même information dans leurs bases de données. Due à cette hétérogénéité, l'intégration ou le simple échange des données n'est pas une tâche facile si les différents intervenants (producteurs ou consommateurs d'information) ne s'entendent pas sur la sémantique des données. Il est donc très difficile de retrouver la sémantique d'un concept dans toutes les bases de données où il peut être représenté.
Dans ce sens, nous avons conçu un système qui s'est focalisé particulièrement sur la récupération et la représentation des différents schémas des bases de données hétérogènes sous forme de metaschéma XML. L'extraction est automatisée et permet au fur et à mesure d'intégrer de nouvelles bases de données sous une forme unique de représentation des concepts sans extraire les données elles-mêmes. Le but du metaschéma ici est de décrire et de porter la sémantique des schémas originels des bases de données à fédérer.

Mots clés
Bases de données hétérogènes, fédération, intégration, langage XSD, metaschéma XML.

Abstract
Databases management systems use various structures and storage formats to represent the same information. Due to this heterogeneity, both data integration and simple data exchange are difficult, if the various actors (data producers and consumers) to not agree on the semantics of the data. It is thus very difficult to find the semantics of a concept in all the databases where it can be represented.
To solve this problem, we conceived a system primarily focused on recovering and modelling multiple heterogeneous databases schema in the form of XML metaschemas.
The extraction is automated and makes it possible to progressively integrate new databases in a uniform concept representation, without extracting the data itself. The metaschema are used to describe and express the semantics of the original schema of the databases to be federated.

Keywords
Heterogeneous databases, federation, integration, XSD language, metaschema XML.

Remerciements

Au terme de ce mémoire, il m'est agréable d'exprimer, en premier lieu, ma profonde gratitude à mon directrice de stage Mme Myriam LAMOLLE pour la confiance qu'elle m'a témoignée en me proposant ce travail. Elle n'a ménagé ni son temps ni sa peine pour me conseiller et m'ouvrir des perspectives nouvelles. Je lui en suis très reconnaissant et lui adresse mes remerciements sincères.

Je tiens à remercier toutes les personnes de l'IUT de Montreuil pour m'avoir accueilli lors de ce stage et permis d'acquérir une expérience très agréable.

Je tiens à remercier plus particulièrement :

Mme Nelly BENSIMON, pour son accueil bienveillant et chaleureux, ainsi que ses encouragements incessants.

Mme Nédra MELLOULI-NAUWYNCK pour m'avoir donné le premier contact avec ce laboratoire.

L'équipe de la salle serveur, pour son aide pendant ce stage.

Mes remerciements s'adressent également aux enseignants qui m'ont fait le grand plaisir d'accepter de juger ce travail. Et aux enseignants de l'Université de Marne-la-Vallée et de l'INSTN, qui ont participé à titres divers dans ma formation de DEA.

TABLES DES MATIERES

TABLE DES ILLUSTRATIONS

Chapitre 1

Introduction

1.1 Contexte

Aujourd'hui différentes organisations doivent faire face au challenge de l'optimisation de l'interopérabilité des systèmes de bases de données afin de pouvoir réaliser des applications critiques [LAU02]. La répartition et le couplage de très grandes bases de données posent des problèmes qui sont, d'une part, étroitement liées à l'hétérogénéité des données, à leur différence sémantique, mais aussi aux différences en termes de fonctions offertes, de disponibilité de coopération (autonomie), et d'accessibilité des sources. Il est donc nécessaire de concevoir des solutions (infrastructures, protocoles d'accès, architectures, etc.) permettant (i) d'offrir en permanence sur le réseau des ressources (bases de données) de manière transparente et (ii) de construire des systèmes de traitement de données sur de grandes quantités de données multiformes stockées dans des bases interconnectées pour des applications telles que la génomique, l'ingénierie des langues ou des documents, les applications scientifiques, le traitement de données financières, etc.

Dans plusieurs systèmes d'informations, les données sont réparties dans des sources différentes, elles-mêmes également réparties dans des machines géographiquement distribuées. Les données peuvent être distribuées au sens relationnel (répartition horizontale et verticale) ou bien dupliquées sans que les réplicas soient structurés de manière homogène. Mis à part les problèmes de gestion associés aux données distribuées, les bénéfices de la distribution sont la disponibilité, la fiabilité et l'amélioration du temps d'accès [VAR02].

L'hétérogénéité, quant à elle, est indépendante de la distribution physique des données. Un système d'informations est homogène si le logiciel qui gère les données est le même sur tous les sites, les données ont le même format et la même structure (modèle de données) et appartiennent à un même univers de discours [AND94]. Un système hétérogène est celui qui n'adhère pas à toutes les caractéristiques d'un système homogène ; c'est-à-dire le système d'informations utilise des langages de programmation et d'interrogation, des modèles, des SGBD différents.

1.2 Problématique et objectifs

L'importance de pouvoir intégrer différentes sources d'information (et notamment des bases de données) au sein d'une organisation n'est plus à démontrer. En effet, les besoins en informations sont de plus en plus nombreux ; les applications, de plus en plus complexes, nécessitent un partage de l'information, devenue une ressource stratégique dont l'utilisation doit être optimisée.

Aussi, la récupération des informations pertinentes devient-elle encore plus difficile quand les données proviennent de sources hétérogènes (bases de données relationnelles, bases de données objets, bases de données XML, etc.). Les données représentant le même concept sont structurées totalement différemment selon le système de stockage. Cependant, il est indispensable de retrouver la sémantique attachée à ces données malgré leurs structurations différentes.

Parmi les travaux de recherche en cours de développement au sein de notre laboratoire, l'intégration de bases de données hétérogènes via XML est un axe de recherche important dans le sens où l'approche XML est générique et utilisable comme support d'autres projets en

cours (par exemple pour les agents conversationnels qui vont rechercher de l'information sur le Web).

Cette intégration à été découpée en plusieurs sous-projets en cours de développement à savoir:

- Développement d'un jeu de tests pour extraire les schémas de différentes bases de données.
- Définition de règles de passage d'un schéma à un autre pour construire automatiquement des métaschémas.
- Définition d'un corpus de métaconnaissances à mémoriser par le métaschéma.
- Mise en place d'opérateurs de transformation basés sur la théorie des catégories.
- Formalisation des requêtes en langage semi-naturel.

Dans ce stage de DEA, nous nous intéressons plus particulièrement au premier point, soit l'extraction et la représentation des différents schémas des bases de données sous forme de schéma XML. Autrement dit, nous construisons une collection de métaschémas XML au dessus des schémas des bases de données à fédérer. Cette fédération reste virtuelle grâce à l'extraction des différents schémas des bases de données correspondantes à fédérer. Notre approche est basée donc sur une architecture générique XML qui permet de gérer l'évolution des schémas des bases de données et la génération de métaschéma XML.

L'association des métaschémas aux bases de données permet de réaliser les finalités suivantes, au travers d'un métaschéma XML:

- Extraction de la sémantique attachée aux données.

- Description complète du schéma de la base de données à partir des catalogues système du SGBD correspondant.

- Optimisation de l'insertion d'informations sur le schéma de la base.

- Meilleure compréhension entre différents types de bases de données et meilleure compréhension des bases sans recourir aux données elles-mêmes.

- Transparence complète des sources, des types et des structures de données.

- Préservation autant que possible du schéma initial de la base de données (mais privilégier les liens vers de nouveaux concepts ou la modification d'un concept) pour éviter la modification de l'information contenue dans les BD.

1.3 Plan

Ce mémoire est structuré de la façon suivante :

Chapitre 2 : **Etat de l'art** – ce chapitre présente en premier lieu la démarche d'intégration des bases de données hétérogènes. Cela permet de voir les différentes étapes qui constituent ce processus. Ensuite, nous abordons les travaux et les projets de recherche sur l'intégration de BD. Cela permet de situer notre approche dans le domaine. A la fin de ce chapitre, nous dressons un bilan qui englobe les principaux objectifs de ces approches, ainsi que leurs avantages et inconvénients.

Chapitre 3 : **Transformation d'un schéma de bases de données en un schéma XML** – lors de ce chapitre, nous présentons notre approche d'intégration. Après la présentation de l'architecture fonctionnelle de notre système, nous définissons les règles de transformation d'un schéma de BD en un schéma XML. Chaque règle est détaillée et illustrée par un exemple d'application qui montre son fonctionnement.

Chapitre 4 : **Méthodologie** – reconstitue la mise en œuvre et également l'implémentation de notre approche. Une modélisation UML pour le noyau du système est utile pour introduire son implémentation. Enfin, nous consacrons la dernière partie de ce chapitre pour la discussion sur les résultats obtenus et notamment les améliorations et les extensions possible.

Chapitre 5 : **Conclusion et perspectives** – ce chapitre analyse les résultats présentés dans le chapitre précédent et évoque un certain nombre de perspectives.

Chapitre 2

Etat de l'art

2.1 Introduction

L'augmentation exponentielle des échanges d'informations par le Web a mis en exergue les problèmes d'hétérogénéité de représentation et de stockage de l'informations.

Dans ce cadre, nous présentons dans ce chapitre les différentes approches d'intégration de bases de données hétérogènes, après avoir présenté rapidement les principales phases du processus d'intégration. A la fin du présent chapitre, nous établissons une synthèse récapitulant les avantages et les limites de chaque approche.

2.2 Processus d'intégration

Les bases de données contiennent des représentations d'objets du monde réel, avec leurs liens et leurs propriétés. L'intégration de bases de données, toutefois, dépasse les représentations, pour considérer en premier lieu ce qui est représenté plutôt que comment il est représenté.

La plupart des approches décomposent le processus d'intégration de schémas de bases de données en plusieurs phases, dont les principales sont [JOU99] :

- Pré-intégration: traduction des schémas source dans un modèle de données commun,
- Comparaison: recherche de correspondances inter-schémas,
- Intégration: création d'un schéma global à partir des schémas source, des correspondances inter-schémas et de règles d'intégration,
- Restructuration: modification des schémas source.

2.2.1 Pré-intégration

Le but de cette phase est d'extraire un maximum d'informations sur les éléments (attributs, classes, méthodes, etc.) composant un schéma, ainsi que sur les relations et les règles existant entre ces éléments.

La pré-intégration est rendue nécessaire à la fois par la relative pauvreté sémantique exprimée par les schémas de bases de données et par l'hétérogénéité des modèles de données utilisés.

2.2.2 Comparaison

Lorsque les schémas initiaux ont atteint le niveau de conformité souhaité, l'étape suivante consiste à identifier les éléments communs des bases existantes.
Rappelons que deux bases de données ont un concept commun si les portions de monde réel qu'elles représentent ont des éléments communs soit une intersection non vide, ou ont des éléments en relation entre eux par un lien sémantique pour des applications futures [PAR96].

Parmi les travaux dédiés à cette phase de comparaison, citons les approches suivantes

- Calcul d'une fonction de similitude entre paires d'éléments [YU91],
- Calcul d'une pertinence sémantique entre éléments à l'aide de la logique floue [FAN91],
- Méthodes manuelles: les paires de noms sont proposées à l'expert de la BD (ou au DBA), qui doit déterminer lui-même les correspondances à retenir.

2.2.3 Fusion

Une fois effectué le travail de description sémantique des schémas (pré-intégration) et de recherche de correspondances entre leurs éléments (comparaison), le processus d'intégration proprement dit peut être effectué. Il consiste, de manière semi-automatique en général, à construire un schéma intégré à partir:

• Des schémas source (éventuellement traduits dans un modèle canonique),
• Des correspondances inter-schémas (déterminées lors de la comparaison),
• De règles d'intégration (propres à la méthode et aux modèles de données utilisés).

2.2.4 Restructuration

Cette phase se retrouve plutôt dans les cas d'intégration de vues ou lorsque la modification ultérieure des schémas source est admise. Les approches récentes, de type fédérées, n'admettent pas de telles modifications, ceci afin de préserver l'environnement local (données, applications) des bases composant la fédération et de garantir leur autonomie [BON94].

Le principe d'intégration a été développé dans divers projets de recherche. Nous allons présenter ci-dessous les travaux en cours prenant en considération les problématiques liés aux échanges de données hétérogènes par le Web.

Rappelons, toutefois, que l'étude faite dans ce mémoire se limite à la phase de pré-intégration.

2.3 Projet GEMO (Ex Verso)

Le projet GEMO [VER02], faisant suite au projet VERSO, consiste en l'intégration de données et de connaissances distribuées sur le Web. L'objectif du projet est l'étude des problèmes fondamentaux posés aux systèmes de gestion de bases de données existants et le développement de solutions novatrices appropriées. Le but est d'obtenir des systèmes plus ouverts à des données complexes et aux réseaux.

Quatre axes de recherche sont en cours pour atteindre cet objectif à savoir :

Médiation entre données XML

Dans le cadre de l'intégration sémantique de données XML, il est important de pouvoir construire aussi automatiquement que possible des ontologies (ou schémas médiateurs) pouvant servir d'interface de requêtes entre des utilisateurs et une collection hétérogène de documents XML. Des correspondances (ou mappings) doivent pouvoir être établies entre l'ontologie servant de schéma médiateur et les différentes DTDs des documents XML relevant de cette ontologie commune.

Des algorithmes de regroupement automatique de données semi-structurées sont à l'étude pour optimiser la construction automatique de schémas médiateurs au dessus d'une collection hétérogène de documents XML. Le but est de regrouper dans des classes des documents XML présentant des similarités, et de calculer pour ces classes les descriptions les plus appropriées (les généralisations les plus précises de l'ensemble des instances de chaque classe) [VER02].

Médiation pour le Web sémantique

L'objectif du Web sémantique est de tendre vers un Web dont la sémantique des données serait à la fois compréhensible par des utilisateurs humains et appréhendable par des entités informatiques (agents, moteurs de recherche, serveurs d'informations)[LAU02]. Le marquage sémantique des données du Web ouvre de nombreuses perspectives d'amélioration de la qualité des moteurs de recherche. Cependant, le passage à l'échelle du Web est un véritable défi qui impose que les problèmes clés soient clairement identifiés et étudiés de manière approfondie en évitant les solutions ad-hoc ne passant pas à l'échelle.

Ouverture vers les services du Web

En combinant les approches de type entrepôt et médiateur, GEMO s'intéresse à intégrer également des services du Web (basés sur l'échange de données XML). Le but est de découvrir des services intéressant pour une application particulière et comprendre comment ils peuvent être utilisés. Dans ce but, des travaux portent en particulier sur un modèle basé sur des documents XML incluant des appels à des services Web.

Entrepôts thématiques de données du Web

Le développement d'une approche flexible et générique permettant de spécifier de façon déclarative les données utiles pour enrichir ou créer un entrepôt thématique est souhaitable en simplifiant l'acquisition à partir du Web, et organiser ces données en vue de faciliter leur interrogation ultérieure. Une première expérimentation basée sur Active XML[1] à commencé [VER02].

Pour cela, GEMO développe en Active XML une bibliothèque de services Web utiles pour la construction d'entrepôts de données thématiques.

1: Active XML (AXML, en bref), un cadre déclaratif pour l'intégration de données, reposant sur les Services Web.

2.4 Projet Xylème

A l'origine, Xylème [XYL03] est une start-up de l'INRIA créée en septembre 2000 pour développer un produit à partir d'un prototype construit dans Verso. Xylème vise à la création d'un entrepôt extensible à très large échelle, capable de stocker toutes les données XML du Web. Elle commercialise cet entrepôt et offre des fonctionnalités d'acquisition de données à partir du Web ou de sources privées, de suivi des changements et d'intégration.
Sa mission est de fournir une nouvelle génération de technologies de gestion du contenu de données, capable d'exploiter le potentiel du langage XML.

L'objectif de ce projet est de permettre aux entreprises d'accéder aux informations du Web aussi facilement que possible. Dans ce cadre, elle fournit des interfaces de programmation (API) permettant de :

- Construire des vues qui intègrent les données du Web par domaine sémantique.
- Interroger les données à travers ces vues grâce à un langage de requête conforme bien sûr aux normes du consortium W3C [XYL03].
- Suivre les évolutions des données d'un domaine (Bibliothèque, Journaux,..).

Bien que la grande majorité des données soient semi-structurées, la plupart des solutions du marché proposent des technologies conçues pour du contenu sans structure (systèmes dits «plein-texte») ou au contraire pour du contenu très structuré (base de données relationnelles).

A l'heure actuelle, un entrepôt natif XML (Xylème Zone Server) gère le stockage, l'accès et à la distribution de contenu semi-structuré hétérogène.

L'efficacité de ce stockage repose d'une part sur la technologie employée par XyStore[1] qui stocke des documents XML comme des arbres logiques et sur la puissance de XyIndex[2] qui indexe absolument toutes les informations tout en réduisant la taille de l'index utilisé.

Le chargement des documents se fait automatiquement via XyLoader[3] et l'indexation est réalisée à la volée.

Finalement, le plus important, c'est tout simplement retrouver la bonne information au bon moment. Toutefois, plus le volume d'informations croît, plus sa localisation devient un enjeu complexe et un facteur de non-productivité. Pour cela XQL, le langage de requête XML associe les capacités sémantiques des systèmes «plein-texte» à la précision des requêtes fortement structurées des bases de données. Cette combinaison peut résoudre le ce problème.

1: XyStore est un entrepôt XML totalement natif. Les documents XML, représentés par des arbres, sont stockés sans aucune altération de leur structure ou hiérarchie intrinsèque.

2 : XyIndex est une technologie d'indexation réellement innovante qui indexe automatiquement tout le contenu des documents ainsi que toutes les balises et les relations qui les unient entre elles.

3: XyLoader est le composant de chargement automatique de documents. Pendant l'importation de contenu dans Xylème Zone Server, nous pouvons simultanément exécuter des requêtes sur ces mêmes données grâce à un agent performant de synchronisation.

L'avantage de cette technique est qu'elle permet une réelle granularité dans la recherche d'informations, une meilleure capacité à formuler tout type de requête pour répondre aux besoins hétérogènes des utilisateurs et enfin donne la possibilité de réutiliser et redistribuer des données précises et pertinentes.

Contrairement à des solutions rigides qui requièrent une connaissance préalable de la structure des documents avant de les intégrer, Xylème Zone Server permet de conserver les hiérarchies entre les éléments inclus dans les documents. Ce sont, entre autres, ces hiérarchies qui font que le contenu a un sens.

Dans un environnement informatique de plus en plus complexe, l'interopérabilité est essentielle pour garantir une réduction des coûts liés à l'intégration de toutes les applications. Xylème Zone Server a été bâti sur une architecture ouverte de façon à laisser la liberté de créer des applications sur l'entrepôt ou d'intégrer celui-ci avec des solutions tierces sans aucun souci de compatibilité.

2.5 Projet CASTOR

CASTOR [CAS03] est un projet OpenSource de mapping objet/relationnel qui permet de rendre pratiquement transparente la gestion de la persistance de modèles objets en base de données. Il permet la mise en correspondance entre bases de données de n'importe quel type et un objet Java [LAM03] ; c'est à dire que chaque attribut est représenté par une classe Java, manipulée par deux opérateurs : Get et Set.

Il est donc nécessaire de faire le lien entre la logique métier implémentée sous forme d'objets et les tables de bases de données. Les concepts utilisés étant très différents, cela peut rapidement devenir complexe.

CASTOR nécessite un fichier de mapping, et évidemment une base de données. En phase de développement, le modèle objet évolue très vite, et il devient rapidement lourd de gérer à la main la synchronisation entre ces différents éléments [CAS03].

2.6 Bilan

Ce rapide tour d'horizon sur les projets et les travaux de recherche en cours, est synthétisé sous forme d'un tableau comparatif entre ces derniers. Ce tableau fait apparaître les principaux objectifs de chaque approche, leurs points forts et les problèmes et les difficultés rencontrés par chacune d'elles.

	Principe/Objectifs	***Points forts***	***Points faibles***
GEMO	- Concevoir un système qui repose sur l'intégration virtuelle de données. - Concevoir et valider des solutions (originales) à la gestion de données complexe dans les systèmes distribués.	- Amélioration de la recherche sur le Web. - Echange de données sur le Web (sources d'informations et services) - Systèmes distribués à grande échelle (nombreuses sources de données, nombreux clients)	- Les applications deviennent de plus en plus complexes (systèmes virtuels, services de gestion de contenu personnels). - Problème de médiation de données (disparité entre la compréhension des données par l'utilisateur et le schéma de la base).
Xylème	- Nouvelle génération de moteur de recherche. - Gestion des évolutions (historique de données). - Hébergement des données et des applications.	- Acquisition des documents à partir du Web ou en local. - Intégration sémantique de données à travers des vues. - Architecture distribuée. - Basé sur le standard XML du W3C.	- Passage à l'échelle (taille, contenu hétérogène, accès multiple). - Gestion des changements (acquisition des données, gestion des versions, requêtes temporelles).
CASTOR	- Mettre des BD relationnelles et objets dans des documents XML et annuaires LDAP.	- Projet open source. - Performance du système de gestion d'objet.	- Evolution très rapide du modèle objet. - Etape supplémentaire d'intégration par la traduction et la mémorisation des sources de données en objets.

2.7 Conclusion

Dans les différents systèmes décrits ci-dessus, l'intégration des bases de données hétérogènes doit passer par une étape essentielle qui est la transformation des données interrogées. Cette transformation dépend de l'architecture même du système de « mapping ». Cela suppose un pré-traitement spécifique sur les données.

Dans ce contexte, notre approche a pour but d'outrepasser le prétraitement en s'intéressant à la sémantique des données quelles que soient leurs formes, autrement dit à la sémantique portée par les structures des bases de données et non pas aux données elles-mêmes.

Chapitre 3

Transformation d'un schéma de base de données en un schéma XML

3.1 Introduction

Nous nous situons à l'origine du processus d'intégration, dans la phase communément appelée pré-intégration. Comme expliqué au chapitre 2.2.1, cette phase a pour but d'analyser les schémas à intégrer afin d'en extraire le contenu sémantique.

Ce chapitre, présente en premier lieu, notre propre approche d'intégration, tout en justifiant et argumentant nos apports. Ensuite, nous définissons et nous détaillons avec des exemples les règles de transformation d'un schéma de BD à un schéma XML, qui nous permettent d'atteindre notre objectif.

3.2 Notre approche

Notre approche vise à proposer une solution aux problèmes d'hétérogénéité des bases de données. L'idée principale est de représenter les schémas sources des BD par les définitions de schémas XML quelque soit le type de ces schémas sources (relationnel, objet/relationnel, objet). L'extraction est automatisée et permet au fur et à mesure d'intégrer de nouvelles bases de données sous une forme unique de représentation des concepts sans extraire les données elles-mêmes. Le but du métaschéma ici est de décrire et de porter la **sémantique** des schémas originels des bases de données à fédérer.

La construction des métaschémas permettent une description complète du schéma de la base à partir des catalogues systèmes du SGBD correspondant. Nous envisageons d'enrichir ce métaschéma en rajoutant des connaissances introduites par les concepteurs des bases de données.

A l'issu d'une première étude sur l'intégration de bases de données fédérées, l'architecture fonctionnelle se présente ainsi :

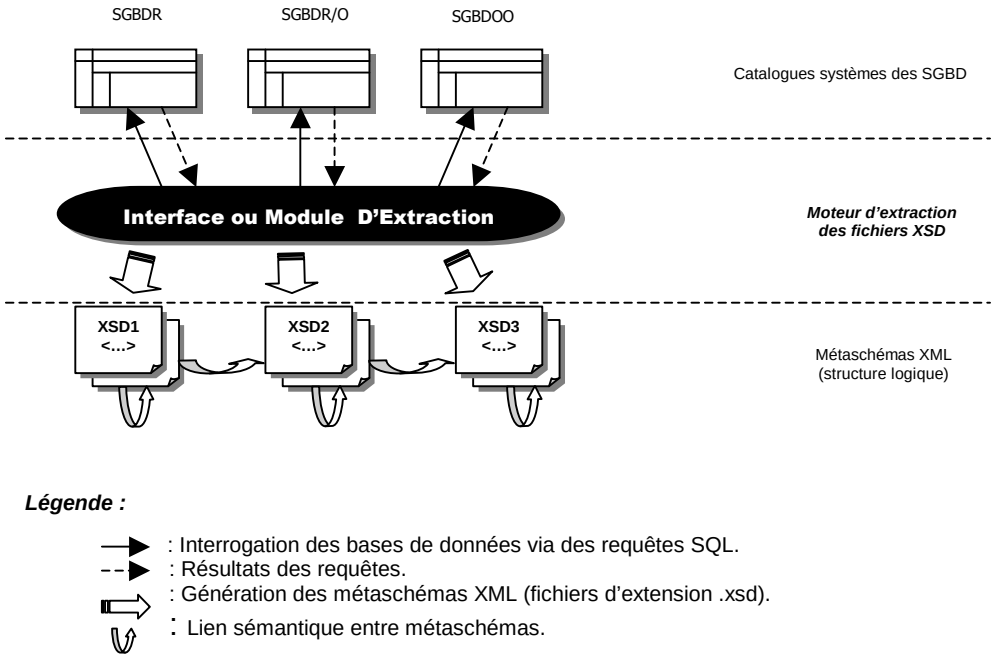


Fig 1. – *Architecture du système.*

Comme nous pouvons le remarquer, le schéma est composé de trois parties principales, dont le rôle de chacune d'elles est explicité ci-dessous, en précisant les relations et les interactions existantes entre ces dernières.

- **Catalogues systèmes des différents SGBD** : afin d'extraire le métaschéma XML, nous devons interroger les différents types de SGBD (relationnel, objet/relationnel, objet) qui contiennent les tables systèmes et également les tables utilisateurs.

- **Module d'extraction** : cette partie constitue le cœur de notre système. A l'aide de cette interface, nous interrogeons les catalogues systèmes sous forme de requêtes SQL et le résultat obtenu par ce module sera transformé sous forme de métaschémas XML.

  Pour la génération des métaschémas XML, notre extracteur repose sur un ensemble de règles de transformation que nous exposerons au chapitre 3.4.

- **Métaschémas XML** : cette partie présente le résultat escompté. Il s'agit des fichiers XSD (plus souple que les DTD comme nous le démontrerons au paragraphe suivant) proprement dit construit par la partie précédente (module d'extraction).
  Donc, chaque type de schéma de BD est traduit en un fichier XSD qui représente sa structure logique et qui est l'objectif principal de notre travail.

Il est à noter que la partie « lien sémantique entre métaschémas » ne rentre pas dans le cadre du présent travail ; elle fait l'objet d'un autre projet de DEA dans notre laboratoire qui s'intéresse à concevoir un modèle permettant de définir les règles de passage d'un métaschéma à un autre.

3.3 Intégration basée sur XML

Pour décrire la sémantique du schéma des bases de données fédérés (voir figure 2), nous nous basons sur la formalisation en XML du métaschéma, sous forme de fichier XSD.

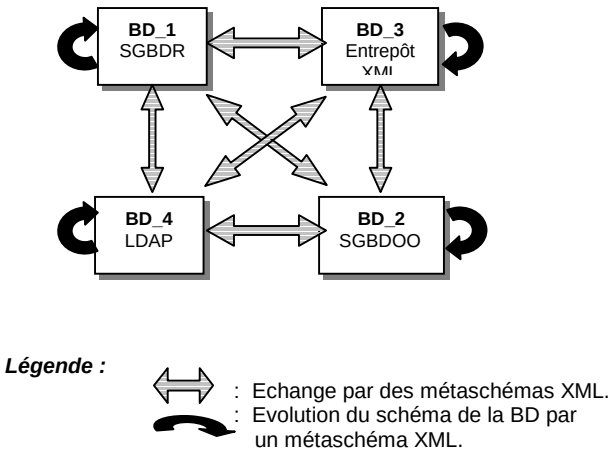


Fig 2. – *Transfert d'informations via XML.*

XML est un langage permettant la description de documents à l'aide de balises porteuses de valeurs sémantiques, il rassemble, en effet, une série de règles permettant de définir ces balises, structurant un document en divers segments identifiables.

Nous avons choisi le langage XML comme formalisme de représentation pour sa capacité à représenter tout type de données. Notamment, le formalisme XSD est beaucoup plus riche que celui des DTD [WIL03] pour représenter la structure permise pour un document, et contrôler de façon plus stricte le contenu pouvant apparaître dans un document XML. Autrement dit, un schéma XSD ou une DTD constituent l'ensemble des règles devant être respectées par tout document XML. Le point faible des DTD est sa faible capacité à spécifier la sémantique des données (ce qui entraîné la naissance du langage XSD). Les DTD sont donc insuffisantes dans le cadre de ce projet car les règles de structure ne sont pas assez précises sémantiquement (pas de typage possible des données, pas de cardinalités précises d'occurrences d'un élément, etc.). Si nous pensons maintenant en terme d'arbre, les DTD ne permettent pas de définir des règles de construction d'arbre XML par héritage et ne sont pas homogènes au niveau du langage avec XML.

En effet, trois types de préoccupations nous poussent aujourd'hui à abandonner les DTD et à rechercher de nouveaux formalismes de modélisation générique de documents XML [CHA03].

- Augmenter la richesse des contraintes exprimables (notamment, pouvoir formuler des contraintes portant sur les contenus et le typage des données) ;

- Tirer partie des formalismes de modélisations de données orientés objet, et introduire des notions telles que l'héritage et la spécialisation de classes, qui permettent de gagner en efficacité et en clarté dans le développement d'un schéma XSD.

- Adopter une syntaxe XML pour représenter le schéma XML lui-même, ce qui permet de réaliser une économie en termes d'outils pour éditer, manipuler et stocker les schémas puisque ceux-ci deviennent eux-mêmes des documents XML et sont interrogeables directement par des langages de requêtes XML tel que XQL.

Le métaschéma XML permet une description complète du schéma de la base de données à partir des catalogues systèmes du SGBD correspondant. Ce dernier sera enrichie en rajoutant des connaissances introduites par les concepteurs des bases de données.

Pour une meilleure compréhension et pour bien éclaircir les points cités précédemments, nous présentons une instance d'un document XML une fois avec sa DTD associée et une autre fois avec son schéma XML correspondant. Le but est de montrer la différence entre ces deux structures et voir concrètement les limites des DTD et les avantages des schémas XML.

La figure suivante présente un court exemple de document XML : il s'agit de la représentation d'un bloc de données structurées véhiculant des informations météorologiques.

```
<?xml version="1.0" encoding="UFT-8"?>
<bulletin-meteo ident="b201">
        <note>&#x00A9; <I>InfoMétéo</I>, </note>
        <date>25 mars 2001</data>
        <heure>08:00</heure>
        <vue-satellite type="gif"  src=http://www.zzz.com/photos/im9245.gif/>
       < !—Localisation géographique -->
        <localisation>
                <zone>
                        <ville>Toulouse</ville>
                        <departement>31</departement>
                        <region>Midi-Pyrénées</region>
                        <pays>France</pays>
                </zone>
                <altitude unite"m">187</altitude>
        </localisation>
        < !—Bloc de mesures -->
        <mesures>
                <ciel>variable</ciel>
                <température unite="celsius">16</temperature>
                <vent>
                        <direction>SW</direction>
                        <vitesse unité="m/s">6</vitesse>
                </vent>
                <pression unite="mb">1025</pression>
                <humidite unite="%">80</humidite>
                <visibilite>10</visibilite>
                <index-uv>1</index-uv>
        </measures>
</bulletin-meteo>
```

Fig 3. – *Extrait d'un simple document XML.*

La figure 4 donne un exemple de DTD qui peut rendre valide le document XML de la figure 3 (Les numéros de lignes ne font pas partie de la DTD).

```
1:  <!DOCTYPE bulletin-meteo [
2:  <!ELEMENT bulletin-meteo (note?, date, heure, vue-satellite?, localisation, mesures)> ← ❶
3:  <!ATTLIST bulletin-meteo  ident ID #REQUIRED> ← ❷
4:  <!ELEMENT note     (#PCDATA | I *>
5:  <!ELEMENT I (#PCDATA)>
6:  <!ELEMENT date     (#PCDATA)>  ← ❸
7:  <!ELEMENT heure (#PCDATA)>  ← ❹
8:  <!ELEMENT vue-satellite  EMPTY>
9:  <!ATTLIST vue-satellite   type CDATA #REQYUIRED
10:                           src CDATA #IMPLIED>
11: <!ELEMENT localisation (zone, altitude)>
12: <!ELEMENT zone (ville, departement, region, pays)>
13: <!ELEMENT mesures (ciel, temperature, vent, pression, humidite, visibilite, index-uv?)>
14: <!ELEMENT ville (#PCDATA)>
15: <!ELEMENT departement     (#PCDATA)>
16: <!ELEMENT region (#PCDATA)>
17: <!ELEMENT pays     (#PCDATA)>
18: <!ELEMENT altitude (#PCDATA)>
19: <!ATTLIST altitude  unite  CDATA #REQUIRED>
20: <!ELEMENT ciel     (#PCDATA)>
21: <!ELEMENT temperature     (#PCDATA)>
22: <!ATTLIST temperature  unite      CDATA #REQUIRED>
23: <!ELEMENT vent (direction, vitesse)>
24: <!ELEMENT pression (#PCDATA)>
25: <!ATTLIST pression  unite CDATA #REQUIRED>
26: <!ELEMENT humidite (#PCDATA)>
27: <!ELEMENT humidite  unite CDATA #REQUIRED)>
28: <!ELEMENT visibilite      (#PCDATA)>
29: <!ELEMENT index-uv (#PCDATA)>
39: <!ELEMENT direction       (#PCDATA)>
31: <!ELEMENT vitesse  (#PCDATA)>
32: <!ATTLIST vitesse unite  CDATA #REQUIRED>
33: ]>
```

Fig 4. – *DTD du document XML.*

❶ Cette ligne comporte un ensemble de déclaration de types d'éléments (<!ELEMENT…). la déclaration de type d'élément indique qu'un élément bulletin-meteo se compose d'une séquence de sous-éléments : la notion de séquence est indiquée par le connecteur « , ». Le connecteur « | » (qui apparaît à la 4ème ligne) indiquerait un choix exclusif entre deux alternatives. Certains de ces sous-éléments, comme l'élément note ou l'élément vue-satellite, sont optionnels, ce que dénote l'indicateur d'occurrence « ? ». Un indicateur d'occurrence « * » (ligne 4) indiquerait qu'un élément ou un groupe d'éléments est optionnel et répétable, « + » qu'il est obligatoire et répétable. La séquence entre parenthèses constitue le modèle de contenu du type d'élément bulletin-meteo. Remarquons qu'il n'est pas possible de spécifier le nombre précis de répétition maximale permise (soit sa cardinalité maximale) d'un élément.

❷ Cette ligne (la 3ème ligne), comporte une déclaration d'attribut (<!ATTLIST…). Elle précise le nom, le type de données et la valeur implicite (le cas échéant) de chaque attribut associé à un type d'élément donné.

❸ Cette ligne (la 6ème ligne), représente la déclaration d'un élément nommé date qui représente la date du bulletin météo. La déclaration du type de cet élément spécifie une donnée textuelle (#PCDATA : Parsable Character Data), ce qui n'est pas le cas en réalité pour notre élément car il doit contenir une date formatée. Ceci est l'un des inconvénients des DTD, puisque des types de données précis ne peuvent être exprimés.

❹ Idem pour l'élément suivant (heure), en principe il doit contenir l'heure du style 15:58 par exemple au lieu d'une chaîne de caractères qui va sûrement modifier le sens et la sémantique du document.

Jusqu'à présent, nous avons utilisé les DTD pour définir et imposer les contraintes à notre documents XML. A travers l'exemple précédent, nous avons déjà vu leurs limites. Dans le fragment suivant nous étudions les possibilités offertes par le langage de schéma XML.

La figure 5 présente un extrait du schéma XML qui correspond à la DTD de la figure 4, en incluant quelques contraintes supplémentaires (en gras dans la figure 5) qui permet d'exprimer ce formalisme.

```
<?xml version="1.0" encoding="UFT-8"?>
<xsd:schema xmlns="http://www.w3.org/2000/10/XMLSchema">
       <xsd:element name="bulletin_météo">
              <xsd:complexType>
                     <xsd:sequence>
                     <xsd:element ref="note" minOccurs="0"/>
                            <xsd:element ref="date" minOccurs="1" maxOccurs="1"/>
                            <xsd:element ref="heure" minOccurs="1" maxOccurs="1"/>
                            <xsd:element ref="vue-satellite" minOccurs="0"/>
                            <xsd:element ref="localisation" minOccurs="1" maxOccurs="1"/>
                            <xsd:element ref="mesures"/>
                     </xsd:sequence>
                     <xsd:attribute name="ident" type="xsd:ID" use="required"/>
              </xsd:complexType>
       </xsd:element>
       <xsd:element name="ciel" type="xsd:string"/>
       <xsd:element name="date" type="xsd:date"/>
       <xsd:element name="direction" type="xsd:string"/>
       <xsd:element name="département" type="xsd:string"/>
       <xsd:element name="heure" type="xsd:time"/>
       <xsd:element name="humidité"/>
              <xsd:complexType>
                     <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                 <xsd:attribute name="unité" type="xsd:string"
                                                                  use="required"/>
                            </xsd:extension>
                     </xsd:simpleContent>
              </xsd:complexType>
       </xsd:element>
</xsd:schema>
```

Fig 5. – *Schéma XML du document XML.*

- Dans le cas d'éléments (dit nœud) terminaux contenant exclusivement des données textuelles et ne portant pas d'attributs (ciel,date,département, etc.), le type d'élément est dit « simple » et sa déclaration associe directement le contenu de l'élément à un type de données, ce type de données peut être un type prédéfini (xsd:string,xsd:date, xsd:time, etc.) ou bien un type « utilisateur » défini ailleurs dans le schéma. Dans notre exemple, la déclaration des types d'éléments date et heure fait référence à des types de données prédéfinis. Ceci permettra à un outil confrontant le document XML à son schéma de vérifier, par exemple, que la forme lexicale du contenu de l'élément date s'interprète bien comme une date.

- Dans les autres cas (le nœud possède des nœuds descendants), le type d'élément est dit « complexe » (xsd:complexType) et son modèle de structure exprime la composition interne de l'élément avec les contraintes imposées (xsd:sequence, etc.) et les degrés de libertés autorisés. La définition explicite des contraintes d'occurrence (minOccurs, maxOccurs) précise les « indicateurs d'occurrence » utilisés dans les DTD. Ils ne sont utilisables qu'au niveau des déclarations d'élément car les attributs de même nom peuvent seulement apparaître une fois dans un élément donnée. Les déclarations d'attributs (xsd:attribute) permettent également d'associer un type de données précis à une valeur d'attribut, une valeur par défaut, s'il est obligatoire ou non.

Si ce court exemple permet d'observer en quoi les schémas offrent des mécanismes de modélisation plus riches et précis que les DTD, au prix d'une syntaxe beaucoup plus verbeuse, il ne permet cependant pas de se faire une idée de toute la puissance qu'apportent les schémas XML en matière de modélisation de données. Notamment, cet exemple illustre bien la possibilité de définir de manière abstraite des types nommés, sortes de prototypes de structures de données que l'on réutilise ensuite dans la déclaration effective de types d'éléments XML (ou d'attributs), moyennant une spécialisation éventuelle par extension ou par restriction (la partie en gras, ligne 24). Une définition abstraite de type peut également être construite par spécialisation en cascade. Cette dissociation possible entre les définitions de types (structurels ou de données) et les déclarations de types d'éléments XML (ou d'attributs) constitutifs d'un vocabulaire traduit l'influence prépondérante des concepts issus de l'orientation objet dans la spécialisation des schémas XML.

3.4 Définition des règles de transformation d'un schéma de BD en un schéma XML

Cette partie concerne les règles de construction retenue pour traduire le schéma d'une base de données sous forme de type XSD. Ces règles tiennent compte de l'aspect statique et dynamique. La partie statique (soit le modèle conceptuel des données) concerne la structure et donc la définition d'un concept. La partie dynamique traduit d'une part les contraintes d'intégrité *(clé primaire, clé étrangère)* et également les contraintes de cohérence *(trigger)* ; d'autre part, elle mémorise les traitements associés aux concepts (soit le modèle conceptuel de traitements).

3.4.1 Traduction de la partie statique

C'est ce qui est représenté par le modèle conceptuel de données (MCD). Chaque entité est la définition d'un concept et chaque association détermine les interactions entre concepts d'où les règles de transformation retenues pour traduire le schéma source d'une base de données en un schéma XSD.

3.4.1.1 Définition des tables de la BD

Règle 1 : Chaque table d'une BD apparaissant dans le catalogue système d'un SGBD est traduite par un type complexe XSD *(<xsd:complexType>)* représentant un concept. Le type XSD est de même facilement traduisible dans un format orienté objet.

3.4.1.2 Définition des attributs non clé d'une table

Règle 2 : Chaque attribut d'une table est traduit en un élément `<xsd:element>` dont le type est un des types de base XSD *(par exemple, `<xsd:string>`)* ou à une restriction d'un type de base (par exemple, une énumération de valeurs ou un intervalle). Dans ce dernier cas, un type dérivé est défini à partir du type de base XSD ; ce type dérivé est alors utilisé pour typer l'attribut de la table.

Lors de la définition de cet attribut, une valeur par défaut peut être définie.

Exemple :

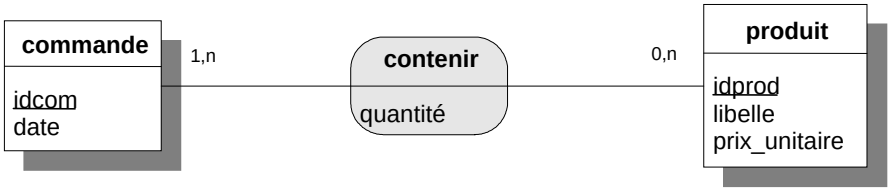
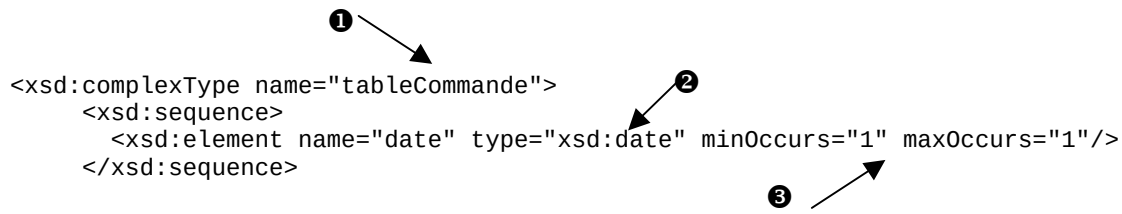


Fig 6. – *MCD de la gestion de commandes.*

Le fragment ci-dessous illustre bien la définition des deux règles de construction précédentes.



❶ le *complexType* est constitué du nom de la table précédé par la mention **`table`** pour tenir compte de la sémantique portée sur le concept.

❷ type de base de la colonne date.

❸ afin de limiter et spécifier les occurrences sur un attribut, nous utilisons les deux attributs, **`minOccurs`** et **`maxOccurs`** qui définissent respectivement le nombre minimal et le nombre maximal d'occurrences de cet attribut (voir annexe).

Soit le modèle relationnel de la BD constitué des tables :

```
COMMANDE [idCom, date]
CONTENIR [#idCom, #idProd, quantite]
PRODUIT [idProd, libelle, prix_unitaire]
```

Dans le modèle ci-dessus, les attributs qui sont soulignés représentent la clé primaire et ceux qui sont précédés par le caractère dièse représentent une clé étrangère.

Le résultat de la transformation des deux entités définissant les concepts de `COMMANDE` et `PRODUIT`, en ce qui concernent les attributs non clé seront représentés par le fragment XSD ci-dessous :

```
<xsd:complexType name="tableCommande">
     <xsd:sequence>
        <xsd:element name="date" type="xsd:date" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
     <... />
 </xsd:complexType>

 <xsd:complexType name="tableProduit">
    <xsd:sequence>
       <xsd:element name="libelle" type="xsd:string"
                    minOccurs="1" maxOccurs="1"/>
       <xsd:element name="prix_unitaire" type="xsd:float"
                    minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <... />
  </xsd:complexType>
```

3.4.2 Traduction de la partie dynamique

3.4.2 Traduction de la partie dynamique des contraintes d'intégrité

3.4.2.1 Contrainte induite par les clés primaires

Dans ce cas, bien souvent, l'attribut de la table portant ce type d'intégrité référentielle n'a pas de valeurs sémantiques par rapport à la définition du concept mais plutôt un moyen technique de déterminer sans ambiguïté une instance du concept. Aussi, la règle générale est-elle :

Règle 3.a : Chaque attribut d'une table faisant partie de la clé primaire est traduit par un attribut sur le type complexe correspondant à la table.

Dans le type complexe `<xsd:complexType name="tableCommande">`, cela correspond à :

```
<xsd:attribute name="idCom" type="xsd:positiveInteger" use="required"/>
```

❶ nom de l'attribut clé de la table **`commande`**.

❷ la restriction **`required`** signifie que la présence de cet attribut est obligatoire.

Nous pouvons interpréter le code comme suit : déclarer un attribut XSD de type entier positif, qui porte le nom `idCom` dont la présence est obligatoire.

Règle 3.b : Dans le type complexe `<xsd :complexeType name= "dbGestion">` (qui représente le complexType de la base entière) et à l'intérieur de la clause

<xsd:element name="commande" type="tableCommande" minOccurs="0" maxOccurs="unbounded">
à laquelle il appartient, la clé primaire est définie comme suit :

```
<xsd:key name="IDCommande">
     <xsd:selector xpath=".//commande" />
     <xsd:field xpath="./@idCom" />
</xsd:key>
```

❶ le nom de la clé primaire sera constitué du nom de la table précédé par **`ID`** afin de le distinguer des autres types de clés.

❷ l'expression **`xpath`** employée récupère le nom de la table concernée par cette clé. Pour cela, nous utilisons l'élément xsd **`<xsd:selection>`** du langage XSL.

❸ Une deuxième expression **`xpath`** est définie avec l'élément **`<xsd:field>`** afin de spécifier le nom de l'attribut clé qui doit être obligatoirement identique au nom de `<xsd:attribute>` du complexType de la table.

Si l'attribut portant la contrainte de clé primaire sert aussi à la définition du concept (souvent le cas des tables à un seul attribut[1] ; par exemple la table `TVA` contenant l'attribut `taux` dont les valeurs possible sont 5.5, 13.8, 33.3) alors cela donnera lieu à un traitement spécifique de mise en forme du type complexe pour définir plus précisément le concept correspondant.

3.4.2.2 Contrainte induite par les associations *(clés étrangères)*

Deux cas sont possibles en fonction des cardinalités indiquées de part et d'autre de l'association.

3.4.2.2.1 Cas des clés étrangères simples [*cardinalité (0,1) ou (1,1)*]

Les cardinalités peuvent s'assimiler à la notion d'agrégation « est une partie de » (*« is_part_of »)* liant deux concepts. La cardinalité (0,1) représente l'agrégation d'un concept dans un autre et la cardinalité (1,1) celle de la composition (où la durée de vie du composant est dépendante de la durée de vie du composé).

Règle 4.a : Soit le concept A (dit contenu) étant une partie du concept B (dit conteneur), B doit alors inclure un attribut qui sera la représentation de la clé étrangère et qui établira le lien entre concept *conteneur* et concept *contenu.*

Exemple : Si nous supposons les hypothèses suivantes dans notre MCD de gestion de commandes, où les contraintes irréelles suivantes ont été exprimées.

- **une commande contient plusieurs produits,**
- **mais un produit est contenu dans une seule commande.**

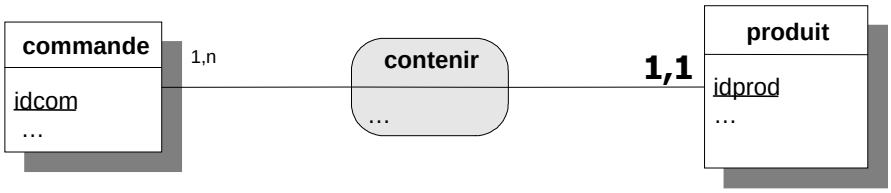


Fig 7. – *MCD de la gestion de commandes avec les nouvelles contraintes.*

1 : Dans ce cas il est possible de faire un type dérivé d'un type de base XSD contenant les valeurs possibles si elles ne sont pas évolutives.

Le résultat de la transformation sera donc pour le conteneur « commande », l'adjonction de la balise `<xsd:keyref>`.

```
<xsd:element name="commande" type="tableCommande" minOccurs="0" maxOccurs="unbounded">
      <xsd:key name="IDCommande">
           <xsd:selector xpath=".//commande" />
           <xsd:field xpath="./@idCom" />
       </xsd:key>

       <xsd:keyref name="IDREFCommande" refer="IDCommande">
           <xsd:selector xpath=".//produit" />
           <xsd:field xpath="./@idCom"/>
       </xsd:keyref>
</xsd:element>
```

❶ ❷ ❸ ❹ ❺

❶ l'attribut xsd **`keyref`** permet la création d'un référence à une clé existante dans le même schéma.

❷ le nom de la clé étrangère est composée de IDREF suivie du nom de la table afin de bien exprimer la notion de clé étrangère.

❸ l'attribut refer de l'élément <xsd:keyref> représente le nom de la clé (key) auquel il est renvoyé.

❹ récupération du nom de la table concernée par cette clé étrangère (la table fille).

❺ spécification du nom de l'attribut qui joue le rôle de la clé primaire dans l'autre table (la table mère).

3.4.2.2.2 Cas des clés étrangères complexes [*cardinalité (0,n) ou (1,n)*]

D'un point de vue sémantique, l'association n'a pas de sens privilégié de lecture. Par exemple, dans le cas de la gestion de commandes, il est aussi pertinent d'énoncer les idées suivantes (voir figure 6) :

- **une commande contient plusieurs produits,**
- **un produit est contenu dans plusieurs commandes.**

Règle 4.b : Nous avons conclu pour exprimer cet état de fait qu'il était préférable de créer un type complexe représentant la notion de « contenir/être contenu » qui n'est pas un concept prépondérant comme peut l'être le concept de *commande* ou de *produit* mais qui permet d'associer les deux derniers cités. En effet traduire la sémantique de « contenir/être contenu » permet de garder deux contextes différents. Dans notre exemple, un technico-commercial va plutôt raisonner en partant du concept de *commande* (pour les ventes) alors qu'un magasinier va plutôt raisonner par rapport au concept de *produit* (pour le réapprovisionnement).

Exemple : Comme dans le modèle relationnel, l'association `contenir` entre `commande` et `produit` sera représentée par une 3ème relation.

Dans le type complexe `<xsd:complexType name="tableContenir">` qui représente également la notion « contenir/être contenu », la clé étrangère sera définit par :

```
<xsd:element name="contenir" type="tableContenir" minOccurs="0" maxOccurs="unbounded">
      <xsd:key name="IDContenir">
           <xsd:selector xpath=".//contenir"/>          ❶
           <xsd:field xpath="./@idCom"/>
           <xsd:field xpath="./@idProd"/>               ❷
      </xsd:key>
```

❶ récupération du nom de la table concernée par cette clé étrangère (la table d'association entre concepts).

❷ spécification des noms des attributs qui jouent le rôle de clé dans les deux autres tables (les concepts)

Vous remarquez ici qu'on a utilisé deux éléments `<xsd:field>` du fait que la clé primaire de cette table est composée de deux attributs. Chacun de ces attributs étant aussi une clé étrangère. Le code XSD ci-dessus interprète la relation `CONTENIR [#idCom, #idProd, quantite]` du modèle logique de la base de données.

Si nous considérons la définition d'un type complexe pour l'association, nous obtenons un graphe des agrégations faibles et fortes des cardinalités exprimés où le sens associé est différent.

Examinons l'exemple de la gestion de commande. Dans ce cas, si nous annulons une commande prise par un client, nous ne supprimons pas les produits contenus dans cette dernière ; au contraire, s'il y a eu une rupture de stock pour un produit désigné, dans ce cas la commande ne peut être établie, et s'il y a des commandes qui contiennent déjà ce produit, elles seront alors annulées.

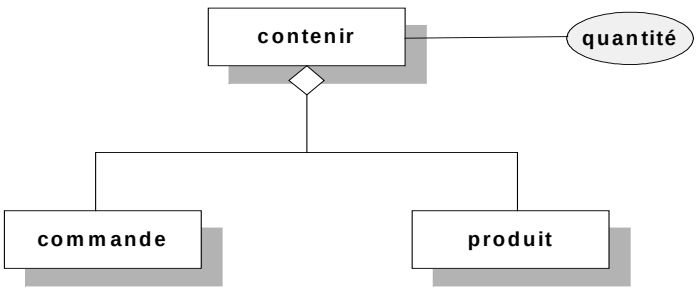


Fig 8. – *Graphe de la gestion de commandes.*

3.4.2.3 Contrainte induite par les contraintes d'intégrité de cohérence de valeurs *(triggers programmés par le DBA)*

Un trigger est un traitement propre à une table et donc au concept associé pour qu'une instance de ce concept reste cohérente tout au long de son cycle de vie.

Il est donc nécessaire de bien mémoriser le trigger dans le type complexe représentant ce concept.
La mise en œuvre de ce mécanisme se fait en plusieurs étapes.

En se basant sur le MCD de gestion de commandes, un trigger nommé `VerifProd` sur la table `produit` se déclenche lors d'une insertion ou d'une mise à jour d'un produit. Il retourne une erreur si le libellé n'est pas renseigné puisque cet attribut doit toujours avoir une valeur.

Voici le code du trigger `VerifProd` dans le cas du SGBD Postgres.

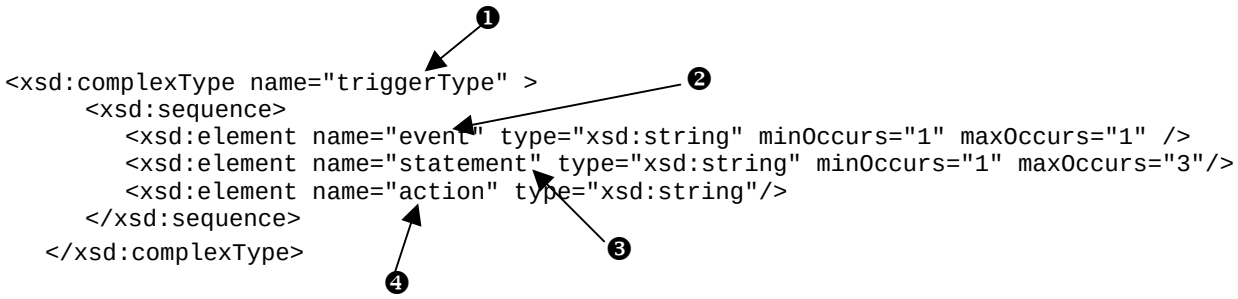
```
create or replace function VerifProd() returns opaque as '
begin
 if NEW.libelle is null then
    raise exception \'Libellé produit doit être renseigné !\';
 end if;
end;
'language 'plpgsql';

create trigger VerifProd
before insert or update on produit
for each row execute procedure VerifProd();
```

Vous remarquez que le trigger est constitué d'une fonction ou procédure qui n'admet aucun argument, et retourne le type `OPAQUE` (type indéterminé). Puis, le trigger fait appel à cette fonction lorsqu'il est déclenché.

3.4.2.3.1 Définition du type trigger

Règle 5.a : Dans un premier temps, nous avons défini un type trigger comme un type de base (selon la norme SQL) en complément des types de bases XSD. Ce nouveau type est de la forme :



❶ le *complexType* d'un trigger est constitué de la mention **`triggerType`** afin de le différencier d'un complexType d'une table ou d'une procédure stockée.

❷ exécution de l'événement: avant, après, et au moment de l'événement, qui sera interprété successivement par les trois clauses suivantes : **`BEFORE | AFTER | INSTEAD`**.

❸ état de l'événement : c'est à dire le moment de déclenchement du trigger, lors d'une insertion, mise à jour, suppression (**`INSERT | UPDATE | DELETE`**)

❹ **`action`** est généralement le corps de la fonction appelée par le trigger et qui est obligatoire pour son exécution.

Nous mémorisons ce nouveau type de bases dans une bibliothèque XSD pour spécifier la norme SQL. Cette bibliothèque constitue une extension du langage XSD et doit être importée en entête de la définition de tout nouveau type de métaschémas XML.

3.4.2.3.2 Utilisation du type trigger

Règle 5.b : Après l'importation de ce nouveau type dans chaque représentation XSD d'une BD, nous extrayons chaque définition de trigger de la base, comme instance du type de base.

Exemple :

```
<xsd:element name="VerifProd" type="triggerType">
        <xsd:sequence>
             <event>before</event>
             <statement>insert</statement>
             <statement>update</statement>
             <action>if NEW.libelle is null then
                        raise exception \'Libelle produit doit être renseigné !\';
                    end if;
             </action>
       </xsd:sequence>
</xsd:element>
```

3.4.2.3.3 Référencement du type trigger dans un type complexe

Règle 5.c : Enfin cette occurrence de ce type trigger est utilisée comme référence (si nécessaire) dans la représentation d'un concept sous forme XSD si la table correspondante dans la BD comporte un (ou des) trigger(s).

```
 <xsd:complexType name="tableProduit">
      <xsd:sequence>                                    ❶
         <xsd:element name="libelle" type="xsd:string"/>
         <xsd:element ref="VerifProd" type="triggerType"/>
       </xsd:sequence>
      <xsd:attribute name="idProd" type="xsd:positiveInteger" use="required"/>
   </xsd:complexType>
```

❶ l'élément **`ref`** permet de référencer la définition du type trigger faites à l'extérieur du type complexe de la table contenant ce trigger.

Au final, le trigger fait partie intégrante du concept extrait de la BD car il traite la cohérence des données liées à ce concept et donc les cas d'exception de ce concept.

3.4.3 Traduction de la partie dynamique de traitements

3.4.3.1 Les procédures stockées

Règle 6 : Les deux premières étapes de construction sont identiques que celle du trigger (définition d'un type, utilisation du type) mais il n'y a pas de référencement dans un type complexe. Cette traduction est faite à l'extérieur de la définition d'un concept (soit un type complexe XSD) car une procédure fait partie du modèle conceptuel de traitements. Elle peut avoir une incidence sur plusieurs concepts dans la mesure où elle va enchaîner une séquence d'actions à réaliser sur une ou plusieurs table de la base de données.

L'ensemble des procédures stockées exprime la façon dont on peut utiliser et gérer la BD.

Exemple :
L'exemple suivant nous représente l'extraction d'une procédure stockée depuis son catalogue système. Grosso modo cette procédure supprime un produit par son identifiant (attribut `idProd`).

```
<xsd:complexType name="del_Prod" type="procType">
    <xsd:sequence>
       <proname>nb_produit</proname>
       <proarg>1</proarg>
       <body>create or replace function del_prod(int) returns int as '
              delete from produit where idProd = $1; '
              language 'sql';
       </body>
    </xsd:sequence>
</xsd:complexType>
```

❶ Chaque procédure stockée est définie par un *complexType*. Le type de ce dernier est **`procType`** (même principe pour les autres règles précédentes).

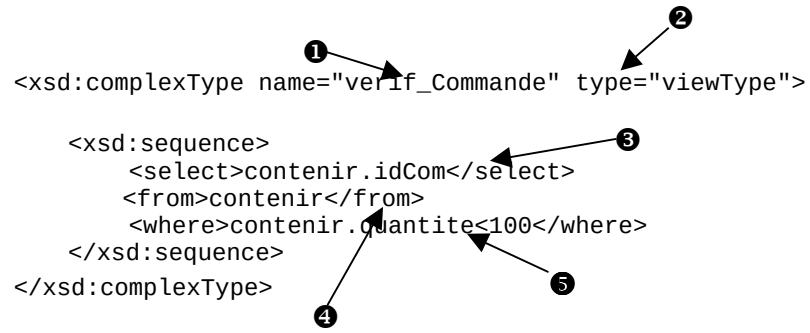
❷ **`proname`** est le nom de la procédure à extraire depuis les catalogues systèmes. Il est de type `string` et chaque procédure doit avoir un et un seul nom.

❸ cet élément est très important pour la définition d'une procédure stockée, car la balise **`<body>`** mémorise le code ou le script de la fonction elle-même.

3.4.3.2 Les vues

Règle 7 : Les vues sont la représentation d'un contexte d'utilisation des concepts contenus dans une base de données ou d'un point de vue à adapter sur ces concepts (vue partielle). Nous préconisons donc de les traduire, comme pour les procédures stockées, à l'extérieur de la définition du type complexe XSD représentant un concept.

Exemple :
L'exemple suivant explique notre méthode d'extraction pour représenter cette partie de BD.
Le rôle de cette vue est d'extraire toute les commandes qui ont une quantité inférieur à 100.



❶ nom de la vue tel qu'il est définit dans la BD.

❷ les vues d'une BD sont du même ordre que les procédures stockées, c'est pour cela qu'on doit définir un type complexe qui porte le type **`viewType`**.

❸ comme la vue d'une BD n'est qu'une simple requête SQL, donc pour sa transformation dans notre fichier XSD, nous avons découpé cette dernière en trois parties principales. Et on fait la correspondance entre les parties de la vue avec les différents éléments XSD créer spécialement. La première partie est la clause **`select`**, son contenu est exporté dans `<xsd:element name="select" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>`

❹ idem pour la seconde partie incluse dans la clause **`from`**.

❺ idem pour la partie condition qui sera transformée dans l'élément XSD **`where`**.

3.5 Récapitulatif

Après avoir défini l'ensemble des règles de transformation d'un schéma de base de données en un schéma XML, nous dressons le tableau suivant, qui résume ces dernières.

Règles de transformation			Exemple résultats de la transformation	Eléments XML Schema	Attributs des éléments XML Schema
Traduction de la partie statique	***REGLE1 :*** *TRANSFORMATION DES TABLES DE LA BASE DE DONNEES*		**<xsd:complexType name**="tableCommande"**>**	complexType	name
	Règle 2 : *Définition des attributs non clé de la table*		*<xsd:element name**="date"** type**="xsd:date"*** **minOccurs**="1" **maxOccurs**="1"**/>**	element	name type minOccurs maxOccurs
Traduction de la partie dynamique des contraintes d'intégrité	***Règle 3 :*** *Définition des clés primaire*	***Règle 3.a :*** Dans le complexType de la table	*<xsd:attribute name**="idCom"*** *type**="xsd:positiveInteger"*** *use**="required"**/>*	attribute	name type use
		Règle 3.b : Dans le complexType de la base	**<xsd:key name**="IDCommande"**>** **<xsd:selector xpath**=".//commande" **/>** *<xsd:field xpath**="./@idCom"** />* **</xsd:key>**	key selector field	name xpath
	Règle 4 : *Définition des clés étrangères*	***Règle 4.a :*** Clés étrangères simples	**<xsd:keyref name**="IDREFCommande" **refer**="IDCommande"**>** **<xsd:selector xpath**=".//produit"**/>** **<xsd:field xpath**="./@idCom"**/>** **</xsd:keyref>**	key selector field	name refer xpath
		Règle 4.b : Clés étrangères complexes	**<xsd:key name**="IDContenir"**>** **<xsd:selector xpath**=".//contenir"**/>** **<xsd:field xpath**="./@idCom"**/>** **<xsd:field xpath**="./@idProd"**/>** **</xsd:key>**	key selector field	name xpath
	Règle 5 : *Définition des triggers*	***Règle5.a :***Définition du type trigger	**<xsd:complexType name**="triggerType" **>** **<xsd:sequence>** **<xsd:element name**="event" **type**="xsd:string" **minOccurs**="1" **maxOccurs**="1" **/>** **<xsd:element name**="statement" **type**="xsd:string" **minOccurs**="1" **maxOccurs**="3"**/>** **<xsd:element name**="action" **type**="xsd:string"**/>** **</xsd:sequence>** **<xsd:complexType>**	complexType sequence element	name type minOccurs maxOccurs
		Règle5.b :Utilisation du type trigger	**<xsd:element name**="VerifProd" **type**="triggerType"**>** **<xsd:sequence>** <event>before</event> <statement>insert</statement> <statement>update</statement> <action>if NEW.libelle is null then raise exception \'Libelle produit doit être renseigné !\'; end if; </action> **</xsd:sequence>** **</xsd:element>**	element sequence	name type
		Règle5.c :Référencement du type trigger	<xsd:element ref**="VerifProd"** type**="triggerType"**/>	element	ref type
Traduction de la partie dynamique de traitements	***Règle 6 :*** *Définition des procédures stockées*		**<xsd:complexType name**="del_Prod **type**="procType" **>** **<xsd:sequence>** <proname>nb_produit</proname> <proarg>1</proarg> <body>create or replace function del_prod(int) returns int as ' delete from produit where idProd = $1; ' language 'sql'; </body> **</xsd:sequence>** **</xsd:complexType>**	compelxType sequence	name
	Règle 7 : *Définition des vues*		**<xsd:complexType name**="verify_commande" **type**="viewType" **>** **<xsd:sequence>** <select>contenir.idCom</select> <from>contenir</from> <where>contenir.quantite<100</where> **</xsd:sequence>** **</xsd:complexType>**	compelxType sequence	name

Remarque : Au court de ce chapitre, nous avons employé le préfixe **`xsd`** devant chaque élément du schéma XML. Ce préfixe est choisi arbitrairement (c'est par convention le suffixe utilisé) qui va permettre d'accéder, dans l'élément **`xsd:schema`**, à tous les éléments et attributs définis dans la syntaxe des schémas.

Voici le fichier XSD « `gestCommande.xsd` » qui représente le MCD de la gestion de commande (voir figure 6) et qui englobe toutes les règles précédentes.

```
 <?xml version="1.0"  encoding="UTF-8" ?>
<!-- edited with XMLSPY v5 rel. 4 (http://www.xmlspy.com) by iut (linc) -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation>
            databaseName: db_gestCommande
             databaseType: Relational
             Other informations
        </xsd:documentation>
    </xsd:annotation>
    <!-- définition of constraint (restriction of type) -->
    <xsd:simpleType name="lib">
        <xsd:restriction base="xsd:string">
            <xsd:minLength value="32"/>
        </xsd:restriction>
    </xsd:simpleType>
    <!--definition of procedure -->
    <xsd:complexType name="del_Prod" type="procType">>
        <xsd:sequence>
            <proname>nb_produit</proname>
            <proarg>1<proarg>
            <body>create or replace function del_prod(int) returns int as '
                    delete from produit where idProd = $1; '
                    language 'sql';
            </body>
        </xsd:sequence>
    </xsd:complexType>
    <!--definition of view -->
    <xsd:complexType name="verify_commande" type="viewType">>
        <xsd:sequence>
            <select>contenir.idCom</select>
            <from>contenir<from>
            <where>contenir.quantite<100</where>
        </xsd:sequence>
    </xsd:complexType>
   <!--definition of trigger -->
    <xsd:element name="VerifProd" type="triggerVerifProd">
        <xsd:sequence>
            <event>before</event>
            <statement>insert</statement>
            <statement>update</statement>
            <action>if NEW.libelle is null then
                        raise exception \'Libelle produit doit être renseigné !\';
                    end if;
            </action>
        </xsd:sequence>
    </xsd:element>
…
```

(Suite du fichier `gestCommande.xsd`*)*

```
<!-- complexType of commande -->
    <xsd:complexType name="tableCommande">
        <xsd:sequence>
            <xsd:element name="date" type="xsd:date"/>
        </xsd:sequence>
        <xsd:attribute name="idCom" type="xsd:positiveInteger" use="required"/>
    </xsd:complexType>
    <!-- complexType of produit -->
    <xsd:complexType name="tableProduit">
        <xsd:sequence>
            <xsd:element name="libelle" type="xsd:string" minOccurs="0"/>
            <xsd:element ref="VerifProd" type="triggerVerifProd"/>
        </xsd:sequence>
        <xsd:attribute name="idProd" type="xsd:positiveInteger" use="required"/>
    </xsd:complexType>
    <!-- complexType of contenir -->
    <xsd:complexType name="tableContenir">
        <xsd:sequence>
            <xsd:element name="quantite" type="xsd:double" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="idCom" type="xsd:positiveInteger" use="required"/>
        <xsd:attribute name="idProd" type="xsd:positiveInteger" use="required"/>
    </xsd:complexType>
    <!-- complexType of database -->
    <xsd:element name="gestion" type="dbGestion"/>
    <xsd:complexType name="dbGestion">
        <xsd:sequence>
            <xsd:element name="commande" type="tableCommande" minOccurs="0" maxOccurs="unbounded">
                <xsd:key name="IDCommande">
                    <xsd:selector xpath=".//commande"/>
                    <xsd:field xpath="./@idCom"/>
                </xsd:key>
                <xsd:keyref name="IDFREFCommande" refer="IDCommande">
                    <xsd:selector xpath=".//contenir"/>
                    <xsd:field xpath="./@idCom"/>
                </xsd:keyref>
            </xsd:element>
            <xsd:element name="contenir" type="tableContenir" maxOccurs="unbounded">
                <xsd:key name="IDContenir">
                    <xsd:selector xpath=".//contenir"/>
                    <xsd:field xpath="./@idCom"/>
                    <xsd:field xpath="./@idProd"/>
                </xsd:key>
            </xsd:element>
            <xsd:element name="produit" type="tableProduit" minOccurs="0" maxOccurs="unbounded">
                <xsd:key name="IDProduit">
                    <xsd:selector xpath=".//Produit"/>
                    <xsd:field xpath="./@idProd"/>
                </xsd:key>
                <xsd:keyref name="IDFREFProduit" refer="IDProduit">
                    <xsd:selector xpath=".//Contenir"/>
                    <xsd:field xpath="./@idProd"/>
                </xsd:keyref>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Fig 9. – *Métaschéma XML de la BD gestion de commandes.*

La figure suivante illustre la représentation partielle graphique obtenue avec XML Spy du type complexe `dbGestion` sous forme d'arbre.

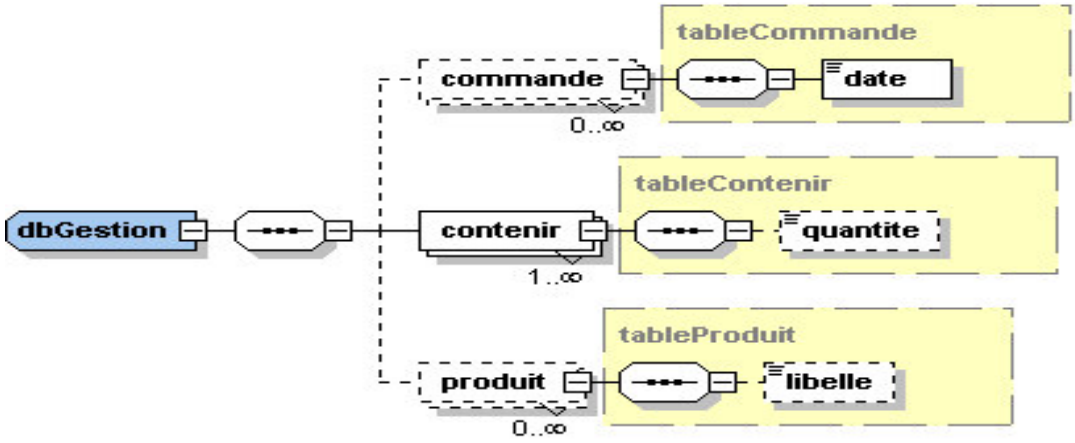


Fig 10. – *Représentation graphique du type complexe* `dbGestion.`

3.6 Conclusion

A l'issu de ce chapitre, nous avons pu, construire l'architecture générale sur laquelle repose notre approche. Nous avons justifié également notre choix d'utiliser XML comme formalisme de représentation des schémas XML pour décrire la sémantique du schéma des bases de données, ainsi que l'ensemble des règles et techniques qui valident notre conception. Dans le prochain chapitre, nous allons présenté le module d'extraction basé sur ces règles et sa mise en œuvre en Java.

Chapitre 4

Méthodologie

4.1 Introduction

Dans le précédent chapitre, nous avons expliqué les règles qui nous ont permis d'extraire les différents schémas des bases de données. Nous nous intéressons maintenant à l'implémentation de notre prototype. Nous présentons également sur sa mise en œuvre ainsi que les problèmes rencontrés lors de cette phase, avec les améliorations et les extensions possible du système.

4.2 Modélisation UML de l'extracteur

Nous proposons une modélisation de notre extracteur, sous forme de diagramme UML. A première vue, nous constatons qu'il représente la structure de données que l'on propose transformer en structure logique XML.

Grâce à la classe Database et à l'aide de ces trois méthodes, nous pouvons extraire les tables, les vues et également les procédure stockées. L'attribut dbmsType est utile du fait qu'il nous renseigne sur le type du SGBD de la base interrogée, et améliore la généricité de notre architecture.

De même, la modélisation des tables, des vues et des procédures stockées donne lieu à trois classes qui dépendent bien sûr de la première classe Database avec des relations d'agrégation. Si nous observons bien la classe Table, nous remarquons qu'elle engendre d'autres structures : les attributs appartenant à une table, les contraintes telles que les clés primaires et clés étrangères, et les contraintes de cohérence (les triggers) qui sont associées à cette table. Pour cela, nous définissons trois classes différentes : les attributs, les contraintes et aussi les triggers.

Pour une extraction générique, nous définissons une classe nommée query. Cette classe représente l'axe de ce diagramme, du fait qu'elle dépend de la plupart des autres classes. Elle possède comme attribut le code de la requête (en langage SQL) et une méthode qui provoque son exécution. Une relation de réflexivité caractérise cette classe dont le but est la récupération du résultat d'une requête par une autre requête.

La figure 11 illustre les différentes relations et les cardinalités entre ces classes sous forme d'un diagramme de classes UML.

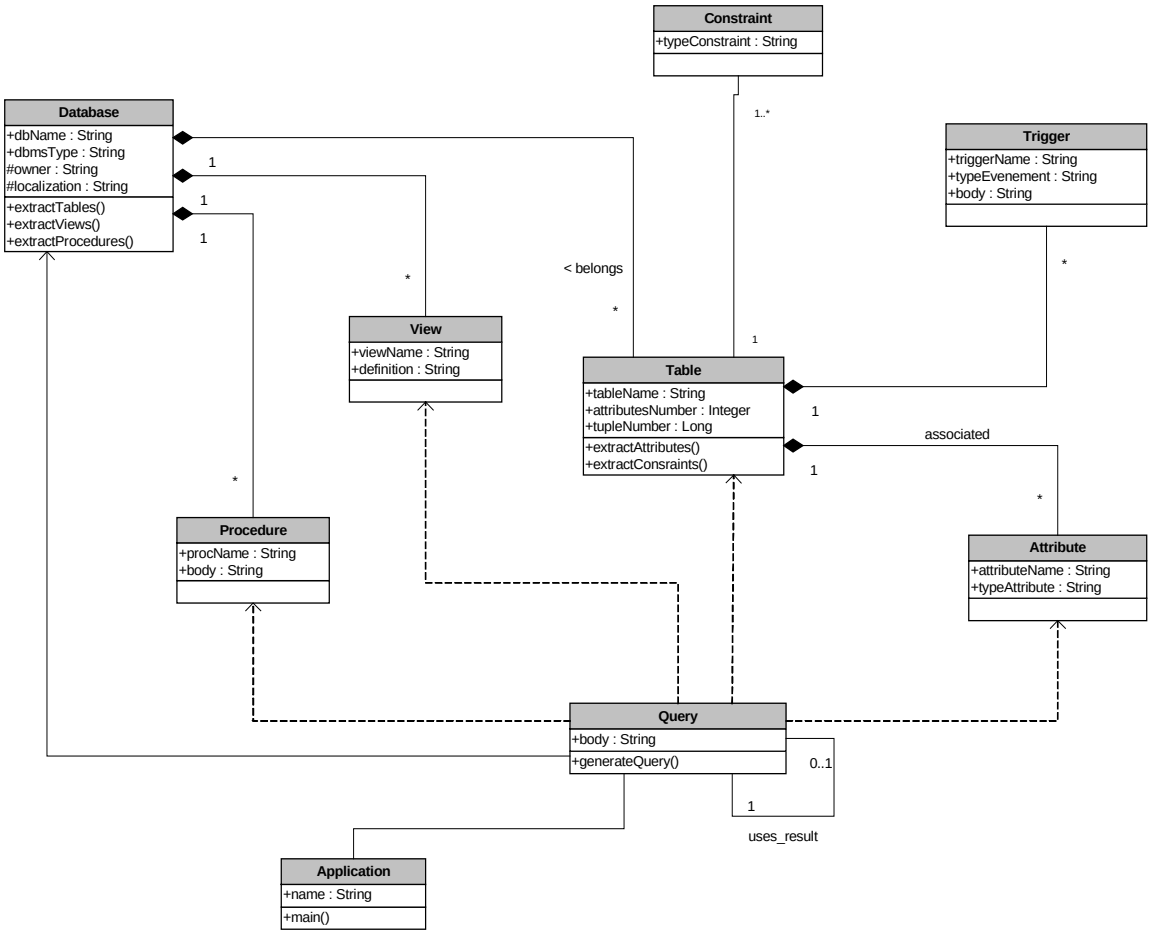


Fig 11. – *Modèle UML de l'extracteur.*

4.3 Mise en œuvre

Nous développons une application Java (langage choisi pour sa portabilité) permettant de prendre en entrée les catalogues systèmes des différents SGBD, de les interroger à l'aide de requêtes SQL et de produire le métaschéma XML correspondant à l'aide de notre modèle.

L'algorithme de transformation comportes trois phases :

- Interrogation des SGBD hétérogènes.
- Récupération des résultats.
- Construction du métaschéma XML.

4.3.1 Interrogation des SGBD hétérogènes

Les différents SGBD contenu dans le système sont interrogeables par notre application à partir de leurs catalogues systèmes (voir glossaire). Nous interrogeons ces tables qui contiennent les différentes bases de données. Dans ce cas, nous formalisons un ensemble de requêtes SQL qui sont génériques et qui peuvent interroger n'importe quel type de bases de données. Cette phase est schématisé dans l'architecture fonctionnelle de notre module (voir chapitre 3.2).

4.3.2 Récupération des résultats

Le résultat récupéré lors de la phase précédente (interrogation des SGBD hétérogènes) représente la structure des bases de données que nous allons coder en XSD. Rappelons que notre but est d'extraire la sémantique portée sur ces bases, c'est-à-dire les tables constituant ces derniers, leurs attributs et leurs types, ainsi que les contraintes et les procédures stockées, les vues et notamment les triggers. Le problème est que toutes ces informations changent d'une base de données à une autre et d'un SGBD à un autre. A titre d'exemple la structure des catalogues systèmes du SGBD Postgres diffère de celui d'Oracle. C'est pour cela qu'il faut une étude préalable pour ce type de tables, afin de savoir exactement l'endroit où nous devons aller chercher l'information, autrement dit paramétrer nos requêtes selon le SGBD.

4.3.3 Construction du métaschéma XML

Après avoir extrait l'ensemble des informations (à partir des schémas des bases de données), nous construisons au fur et à mesure chaque partie du fichier XSD. Pour cela et à l'aide de la bibliothèque Java, nous utilisons l'API DOM pour la construction de l'arbre du document XML et sa hiérarchie (au moment de la rédaction de ce mémoire, il n'existe pas une API Java propre à la structure des schémas XSD). Une autre API nécessaire pour atteindre cet objectif est JAXP, qui est utilisée pour les transformations et l'analyse des documents XML. Enfin, en se basant toujours sur les règles de transformation défini au troisième chapitre (voir section 3.2).
Nous allons voir maintenant en détail comment notre extracteur fonctionne pour construire juste une petite partie du métaschéma XML (construction du type complexe correspondant à une table de la bases de données).

La figure 12 montre le rôle des API DOM et JAXP dans l'extracteur, les sens des échanges d'informations avec les SGBD et la sortie XSD formée par l'extracteur.

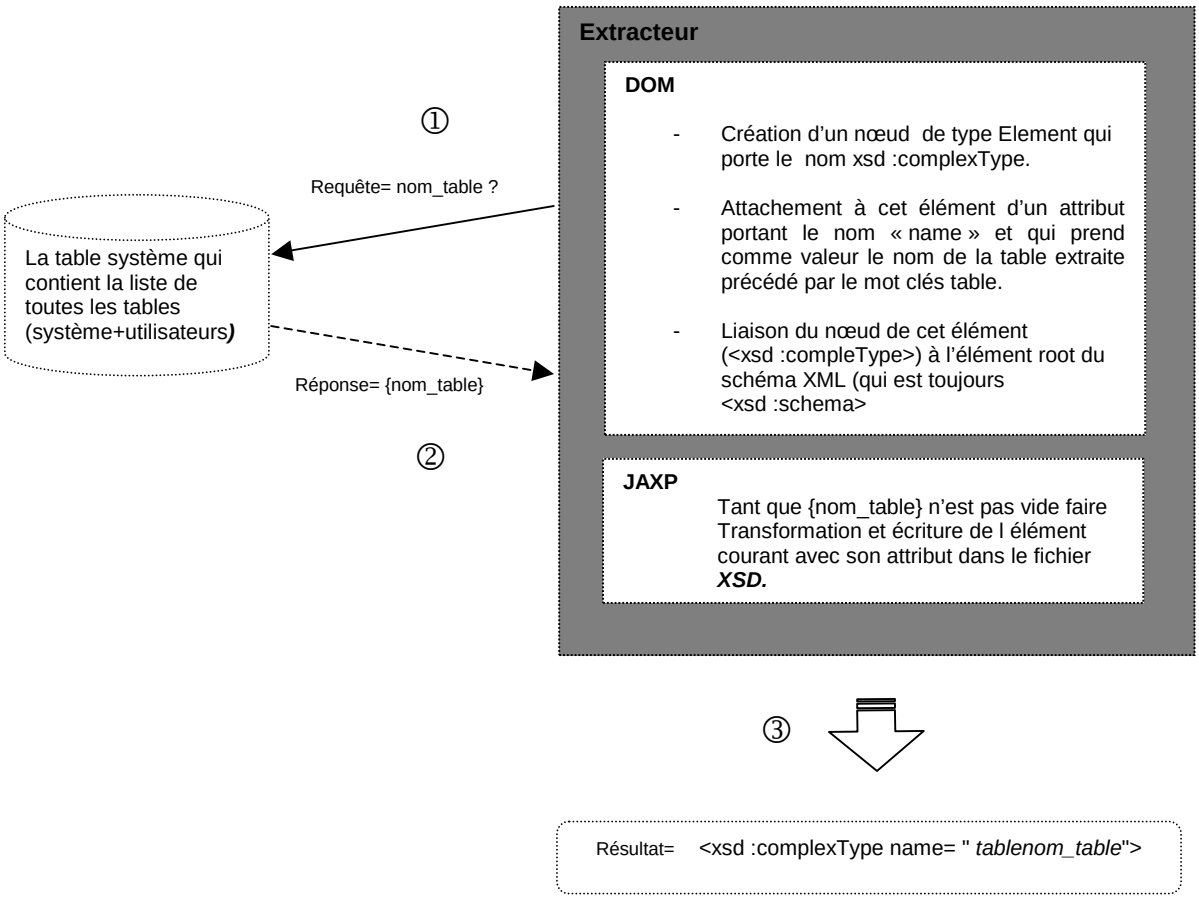


Fig 12.– *Construction d'un complexType représentant une table.*

Trois phases se répètent autant que nécessaire pour extraire le schéma de la base de données à savoir :

① Interrogation des catalogues système sous forme de requête SQL.
② Récupération de l'information (nom de la table).
③ Formalisation du résultat (construction du type complexe de la table).

Avant d'entamer l'implémentation de notre système, voici un algorithme pragmatique (voir figure 13) qui donne une idée générale de son fonctionnement.

```
Extracteur {
Debut

        Choisir_sgbd ;
        Choisir_db ;
        Etablir_connection_db ;

        Si  db  ∃  alors
             {
                Tq  trigger  ∃   faire
                {
                        pour chaque trigger trouvé faire
                        {
                                extraire_nom_trigger ;
                                extraire-event_trigger ;
                                extraire_code_trigger ;
                        }
                        ecrire_fichier_xsd;
                } // extraction terminée des triggers

                Tq   procedure   ∃   faire
                {
                        pour chaque procedure trouvée faire
                        {
                                extraire_nom_procedure ;
                                extraire_corps_procedure ;
                        }
                        ecrire_fichier_xsd ;
                }  // extraction terminée des procedures stockées

                Tq  vue  ∃   faire
                {
                        pour chaque vue trouvée faire
                        {
                                extraire_nom_vue ;
                                extraire_definition_vue ;
                        }
                        ecrire_fichier_xsd ;
                }  // extraction terminée des vues

                Tq  table  ∃  faire
                {
                        pour chaque table trouvée faire
                        {
                                extraire_nom_table ;
                                extraire_attributes ;
                                extraire_type_attributes ;
                        }
                        ecrire_fichier_xsd ;
                }  // extraction terminée des tables

                // définition du complexType de la base
                pour chaque table de la base faire
                {
                        extraire_cle_primaire ;
                        si cle_etrangere  ∃   alors   extraire_cle_etrangere ;
                        ecrire_fichier_xsd ;
                }
        } // fin si

        sinon aller au debut

Fin
}  // fin Extracteur
```

Fig 13.– *Algorithme d'extraction du schéma de la BD.*

4.4 Implémentation

Actuellement, notre prototype est capable d'extraire les schémas des bases de données hétérogènes de type relationnel/objet. Nous envisageons d'étendre le prototype pour traiter les base objet, XML native et annuaire LDAP. Dans notre algorithme nous avons tenu compte de ces types de bases de données, du fait qu'il est conçu d'une façon générique afin de supporter ces types de bases de données.

Notre algorithme d'extraction procède à l'écriture de balises dans des fichiers (auquel on donne une extension « .xsd ») pour fournir en sortie un document XSD. Notre algorithme ne profite donc pas de la structure interne du document XML (arborescence d'élément). Il serait intéressant d'utiliser des bibliothèques Java (Document Object Modélisation) qui reconnaissent le type de données XML mais malheureusement pas le type de données XSD (Fig12) et permettant de manipuler un schéma XML en tant qu'objet. Cette modélisation objet va permettre d'instancier le contenu des balises en accédant au feuilles et au racines de l'objet XML.

```
class DomDocument
       properties :
              version
              encoding
              standalone
              type
       methods :
              root()
              children()
              add_root($node)
              dtd
              dumpmem()
```

Modélisation DOM d'un document XML

```
class DomNode
       properties :
              name
              content
              type
       methods :
              lastchild()
              children()
              parent()
              new_child($name, $content)
              getattr($name)
              setattr($name, $value)
              attributes()
```

Modélisation DOM d'un noeud Dans un document XML

Fig 14. – *Classe DomDocuement et DomNode de la bibliothèque DOM de Java.*

4.4.1 Amélioration des résultats

4.4.1.1 Extraction des informations des BD hétérogènes

Pour toute base de données interrogée par notre extracteur, notre algorithme n'extrait que le schéma de la base, c'est-à-dire sa structure. Alors qu'il peut bien extraire aussi les données et les informations constituant cette dernière. Donc il serait plus intéressant d'extraire aussi les données de la BD elles-mêmes dont le but est de mieux représenter la base de données.

4.4.1.2 Gestion des schémas de bases de données

La génération des métaschémas peut être obtenue selon deux cas. Le premier cas correspond à l'intégration d'une nouvelle base de données dans la fédération de bases de données existantes (cas de notre système). Ceci donne lieu à la création d'un nouveau fichier XSD (issue de la réponse oui de la figure 15).

Le second cas qui n'est pas implémenté par notre algorithme à l'heure actuelle, correspond à l'évolution du schéma d'une base de données déjà intégrée dans une fédération de bases de données. Nous opérons alors une mise à jour du fichier XSD existant (issue de la réponse non de la figure 15.). Les différentes évolutions de schéma sont gardées par un stockage de tous ces fichiers de modification. Ceci nous permet de garder la trace des évolutions successives (versioning) d'un schéma de BD.

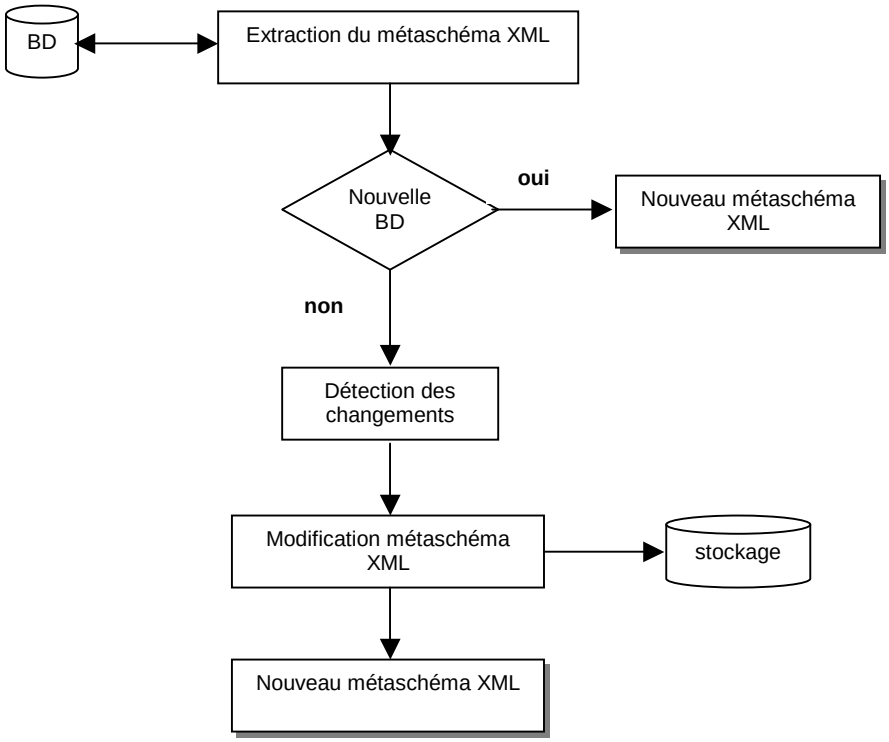


Fig 15. – *Génération des métaschémas XML.*

4.5 Conclusion

Nous avons montré d'une manière assez simple et claire, les grandes lignes de notre implémentation, pour le développement de notre système. Quelques points restent encore à améliorer (par une API JavaDOM spécialement dédié au format XSD), d'autres à développer (évolution de schéma) dans un travail perspectif.

Chapitre 5

Conclusion et perspectives

5.1 Conclusion

Dans ce mémoire, nous avons abordé la problématique de l'intégration de bases de données hétérogènes. Cette hétérogénéité est certainement une tâche complexe. C'est pourtant une tâche que les entreprises peuvent difficilement éviter si elle veulent mettre en route de nouvelles applications (application Web, application spécifique à un domaine, etc.) pour une meilleure productivité.

L'objectif de ce travail est de représenter ces différentes bases de données hétérogènes sous forme de métaschémas XML. Pour cela, deux étapes sont nécessaire à savoir :

- Récupération des schémas des bases de différents SGBD.
- Couplage en une structure logique XML.

Le but, ici est de réaliser une fédération par une intégration de ces bases de données. Notre approche est générique, elle prend en compte l'échange de données et de métadonnées au travers de règles XML dans la perspective de bases de données.

Notre choix des langages XML, XSD a été amplement justifié dans les précédents chapitres où nous avons souligné les avantages et les apports d'une telle technique, notamment dans les architecture distribuées.

Pour la réalisation de notre système, nous nous somme intéressés à deux points. En premier lieu, nous avons consulté et étudié soigneusement les catalogues systèmes des différents SGBD, qui sont eux aussi hétérogènes (au niveau de la conception et de la structure), afin d'extraire la structure des bases de données à intégré. Pour cela nous avons formalisé l'ensemble des requêtes génériques, qui vont interrogées ces catalogues.

Dans un deuxième temps et après l'extraction des schémas, une étape de transformation est indispensable pour la génération des métaschémas. A cet égard, nous nous sommes basés sur les règles de transformation présentées dans le chapitre 3 du présent document, et également sur les interfaces de programmation DOM et JAXP de la bibliothèque Java afin d'obtenir la structure d'arbre hiérarchique du document XML.

Il va de soi que notre système, sous sa forme actuelle, répond aux préoccupations de son utilisateur et lui offre l'accès à une nouvelle structure de données, tout en gardant la sémantique de ces données.

5.2 Perspectives

Les perspectives ouvertes par ce travail sont à moyen terme :

- Des extensions pourraient être apportées, notamment en ce qui concerne l'évolution du schéma d'une base de données déjà intégré dans la fédération. Nous opérons alors une mise à jour du métaschéma XML et un stockage de toutes les modifications apportés à ces derniers.

- Etablissement des correspondances entre ces métaschémas générés, à l'aide de la conception d'un modèle permettant de définir les règles de passage d'un métaschéma

à un autre, constituant ainsi la construction d'une ontologie. Cependant, l'établissement de ces liaisons ne peut être fait sans l'intervention et la collaboration des différents experts des différentes bases de données (traitement semi-automatique).

En fin, les travaux futurs à réaliser sont l'adjonction de connaissances dans les métaschémas, puis la génération semi-automatique des correspondances entre métaschémas, par des fonctions exprimées en XSL permettant de retrouver plus aisément les différentes représentations d'un concept dans la fédération lors de l'intégration de cette dernière par un utilisateur.

Bibliographies

[AND94] M. Anderson. *Extracting an entity relationship schema from a relational database through reverse engineering*, Database laboratory, department of computer science Swiss federal institute of technology, CH-1O15 Lausanne, Switzerland.

Cet article présente une méthode pour extraire un schéma conceptuel à partir d'une base de données relationnelle. La méthode s'est basée sur l'analyse et la manipulation de données dans le code d'une application utilisant un système de gestion de bases de données relationnelle.

[BON94] M. Bonjour, G. Falquet, M. Léonard. *Intégration de concepts et intégration de bases de données*, Actes du X^{ème} congrès INFORSID, Aix en provence 1994.

Cet article aborde le processus d'intégration de schémas. L'approche abordée ici est la construction d'une base de concept à partir de chaque schéma, en utilisant les techniques terminologiques, contrairement aux approche classiques qui proposent d'utiliser un modèle de données canonique. Dans ce cas, le système sera basé sur une architecture à trois niveaux (terminologie, caractères définitoires, représentation dans les schémas).

[BIR01] P.V. Biron, A. Malhotra. *XML Schema Part 2: Datatypes*, W3C Recommendation, 02 Mai 2001, http://www.w3c.org/TR/XmlSchema-0/.

Le site officiel du consortium W3C. XML schemas Part 2 : Datatypes (document concernant les types de données).

[BOU00] Y. Bourda, M. Hélier. Métadonnées, *RDF et documents pédagogiques*, Cahiers GUTenberg n°35-36 – Congrès GUTenberg 2000, Toulouse, Mai 2000.

Cet article présente et définit la notion de métadonnées, son auteur insiste sur la nécessité d'une normalisation, c'est-à-dire pour un même ensemble de métadonnées, on peut trouver une implémentation en XML. Une façon d'obtenir une implémentation unique est d'utiliser RDF. Le Dublin Core et les « métadonnées pédagogiques », dont la normalisation par les IEEE est en cours d'élaboration.

[CAS03] Castor. http://castor.exolab.org

Le site officiel du projet Castor.

[CHA03] F. Chahuneau. *XML : une méthode universelle de représentation textuelle de données structurées*, Support de cours DEA IST, (CEA – INSTN), 2003.

Support de cours proposé au DEA Information Scientifique et Technique dans l'unité : « Normes de représentation dans les systèmes multimédias », qui présente le standard XML comme une méthode universelle de représentation textuelle de données structurées, ainsi que les autres normes standard associés à XML ou dérivés de XML tels que XSL, XPATH, XQuery, etc.

[CHA01] G. Chazalon. *XML schema W3C*, http://www.w3.org/TR/xmlschema-0/, Mai 2001.

Le site officiel du consortium W3C. La recommandation XML Schema.

[DEL01] C. Delcroix. *Plan de transformation d'un schéma conceptuel en un schéma XML*, Facultés universitaires Notre-dame de la paix, Namur Belgium.

Ce document propose un plan de transformation d'un schéma conceptuel en un schéma XML, représentant une DTD. Le plan de transformation comporte neuf étapes. Pour chaque étape, l'auteur précise l'objectif poursuivi et propose un traitement permettant d'y parvenir.

[FAL01] D.C. Fallside. *XML Schema Part 0 : Primer*, W3C Recommendation, 02 Mai 2001, http://www.w3c.org/TR/XmlSchema-0/.

Le site officiel du consortium W3C. XML schemas Part 0 : Primer (Introduction).

[JOU01] F. Jouanot, C. Parents, S. Stefano Spaccapietra. *Intégration de BD et interopérations*, LBD (Laboratoire de Bases de Données).

Ce document est un cours de bases de données sous forme de diapositives qui aborde quelques notions de bases pour éliminer l'hétérogénéité des bases de données.

[JAV03] Java, http://java.sun.com

Le site officiel de java sun microsystem.

[LAL03] Latruiste, http://www.laltruiste.com

hébergé par EuropeanServers, ce site représente un guide pratique pour les langages Web tels que le langages HTML, XHTML, XML, XLL, PHP, etc. Laltruiste propose un apprentissage très simple et efficace pour la création et la manipulation des sites Web.

[LAM03] M. Lamolle, N. Mellouli. *Intégration de bases de données hétérogènes via XML*, EGC'2003, Atelier fouille de données, Lyon 2003.

Cet article, présente une approche générique pour gérer des bases de données hétérogènes via XML. L'architecture proposée ici permet de prendre en compte l'évolution des schémas des bases de données par des métaschémas et de gérer la fédération de bases de données par la mise en correspondance via des fichiers XSL.

[LAU02] P.Laublet, C.Reynaud, J. Charlet. *Sur quelques aspects du Web sémantique*, Actes des deuxièmes assises nationales du GdRI3, 2002.

Comme son titre l'indique, cet article publié en 2002, traite le sujet sur le Web sémantique et ces langages, les métadonnées et la norme RDF. A la fin il aborde le problème d'intégration de sources d'informations .

[MIN01] S.Miniaoui, J. Darmant, O. Boussaïd. *Web data modeling for integration data warehouses*, International workshop on multimedia data and document engineering (MDDE01), Lyon, Juillet 2001.

Ce mémoire de DEA a donné lieu à une publication. Le but de ce mémoire est de concevoir un modèle conceptuel d'un objet complexe qui généralise les différentes données multiformes présentes sur le Web qui sont intéressantes à intégrer dans un entrepôt de données en tant que sources externes.

[MON02] Le monde informatique. *XML et les bases de données : le duel intégral*, N°958 (8 Novembre 2002), pp 16-22.

Ce document est un article sur les bases de données natives XML, sortie dans ce numéro, ils montrent l' utilité et le rôles de ces derniers dans les différents domaines de l'informatique et leurs relations avec les différents SGBD.

[MCL01] B. McLaurhlin. *Java et XML*, Editions O'Reilly, Mars 2001.

Après une introduction rapide à XML, cet ouvrage couvre tout ce qu'un programmeur Java doit connaître pour manipuler des données XML.

[PAR96] C. Parent, S. Spaccapietra. *Intégration de bases de données : Panorama des problèmes et des approches*, Ecole polytechnique fédérale de Lausanne, ingénierie des systèmes d'informations Vol.4, N°3, 1996.

Cet article fournit une vision globale des problèmes soulevés et des approches qui ont été proposés pour réaliser l'intégration de bases de données hétérogènes.

[THO03] J. Thomasson. *Schémas XML*, Groupe Eyrolles 2003.

Ce document propose un tour d'horizon assez complet du rôle et des grandes fonctionnalités de XML schéma. Après avoir expliqué le positionnement de cette nouvelle recommandation du W3C par rapport à ses consoeurs, il présente un aperçu de son vocabulaire spécifique et de ses fonctions de base.

[THO01] S. Thompson, D. Beech, M. Maloney, N. Mendelsohn. *XML Schema Part 1: Structures*; W3C Recommendation, 02 Mai 2001, http://www.w3c.org/TR/xmlschema-1/.

Le site officiel du consortium W3C. XML schemas Part 1 : Structures (document relatif aux structures).

[VID01] C. Vidal. *Tutorial DOM et JDOM*, cyril@planetexml.com

Ce tutorial se donne pour objectif un aperçu des API DOM et JDOM, dont la finalité est de lire et de manipuler des fichiers XML. Il propose un minimum de connaissance à propos du XML et surtout du langage JAVA, exclusivement utilisé ici.

[VER02] Projet VERSO. *rapport d'activité scientifique 2002*, thème INRIA 3A, 2002

Ce document est le récent et le dernier rapport d'activité concernant le projet GEMO. GEMO qui est le successeur du projet VERSO, quand a abordé dans le chapitre état de l'art du présent mémoire.

[VAR02] G. Vargas Solar, A. Doucet. *Médiation de données: solutions et problèmes ouverts*, *Actes des deuxièmes assises nationales du GdRI3, 2002.*

Cet article fait un bilan des challenges introduits par la médiation de données distribuées et hétérogènes. Les techniques existantes apportées au long des quinze dernières années sont brièvement présentées. Enfin, les directions de recherche en termes de problèmes ouverts de la médiation de données sont discutées.

[WIL01] K. Williams & al. *XML et les bases de données*, Editions Eyrolles, 2001.

Ce livre indique comment intégrer XML dans les stratégies actuelles de sources de données relationnelles. Ensuite, il se penche sur les similitudes et les différences entre la conception d'une base de données relationnelles et la conception XML. Il détaille certains algorithmes de transfert de données entre les deux.

[XYL03] Xylème. http://www.xyleme.fr

Le site officiel du projet Xylème.

GLOSSAIRE

Afin de bien exploiter notre glossaire, nous le décomposons en de deux grand thèmes, le premier celui de la galaxie XML et toutes les techniques qui tournent autour, le second est les bases de données, où on trouve également l'ensemble des mots reliés ce dernier.

G.1. XML

Attribut XML [*Attribute*]

Les attributs XML permettent de définir les caractéristiques d'un élément par des couples « nom/valeur ». Par exemple, si l'on considère que les deux caractéristiques (attributs) d'un livre sont le titre et l'auteur, un élément `Livre` peut être écrit en XML sous la forme `<Livre titre="paroles" auteur="prevet"/>`.
Voir aussi : élément XML.

Analyseur XML [*XML parser*]

API analysant et décodant les balises d'un document XML afin de permettre à l'application utilisant cet analyseur de traiter facilement des données au format XML.

Balise [*Tag*]

Une balise permet de délimiter des données dans les langages de balisages tels que HTML, XML, …

CASTOR [*Castor*]

CASTOR est un projet OpenSource de mapping objet/relationnel.
Référence : http://castor.exolab.org

DOM [*Document Object Model*]

C'est une spécification du W3C pour proposer une API qui permet de modéliser, de parcourir et de manipuler un document XML. Le principal rôle de DOM est de fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (parcours, recherche et mise à jour). A partir de cette représentation (le modèle), DOM propose de parcourir le document mais aussi de pouvoir le modifier. Ce dernier aspect est l'un des aspects les plus intéressant de DOM.
Référence : http://www.w3.org/DOM/

Elément XML [*Element*]

Un élément XML est la brique de base d'un document XML. Un élément est composé :
- D'une balise de début contenant des attributs et leurs valeurs,
- De données,
- D'une balise de fin.

GEMO [*Integration of data and knowledge distributed over the Web*]

GEMO consiste en l'intégration de données et de connaissances distribuées sur le Web.
Voir aussi chapitre 2.3

Interopérabilité [*Interoperability*]

L'interopérabilité implique la possibilité de pouvoir demander et recevoir des services entre des « systèmes intéropérables » et pouvoir utiliser leurs fonctions.

JAXP [*Java Api for XML Parsing*]

JAXP est une API conçu par sun microsystem pour le traitement des documents XML en utilisant DOM, SAX et XSLT. JAXP permet à des applications d'analyser et transformer des documents XML.

Voir aussi : analyseur XML.

Métadonnées [*metadata*]

Les métadonnées sont des données structurées, standardisées qui décrivent le contenu de document que souhaite partager des utilisateurs. Ce sont des données qui renseigne sur la nature de certains autres données et qui permet ainsi leur utilisation pertinente.

Métaschéma [*metaschema*]

Un métaschéma XML est un schéma porteur de méconnaissances (sémantique) adjoint au schéma de la base de données. Dans ce document un métaschéma XML est un fichier XSD résultant de l'opération de mapping d'un schéma de base de données.

Namespaces [*espaces de noms*]

Les namespaces permettent de créer des sous ensembles parmi les éléments qui composent un document XML. Tous les éléments appartenant à un namespace sont préfixés par son identifiant suivi du signe deux-points. Ce système permet de lever toute ambiguïté en fournissant des identifiants uniques. Il correspond à la notion de paquetage des langages orientés objets.

Référence : http://www.w3.org/TR/REC-xml-names/

SAX [*Simple Api for XML*]

L'API SAX propose un ensemble standardisé de mécanismes pour manipuler un document XML. Elle fonctionne sur le principe de l'envoi d'événements.

Référence : http://www.saxproject.org/ (le site officiel).

Voir aussi : DOM.

Schémas XML [*XML schemas*]

La spécification XML Schema est un nouveau mécanisme sur lequel travaille le W3C afin de définir des vocabulaires XML en lieu et place des DTD. En conséquence, la spécification XML schémas du W3C se compose de trois parties :

- XML schemas Part 0 : Primer (Introduction),
- XML schemas Part 1 : Structures (document relatif aux structures),
- XML schemas Part 2 : Datatypes (document concernant les types de données).

Voir aussi la partie annexe.

W3C [*World Wide Web Consortium*]

Organisme de proposition et de normalisation des technologies, protocoles et langages du Web. Le site officiel du W3C est http://www.W3.org

Web sémantique [*Semantic Web*]

Le Web sémantique est une nouvelle infrastructure devant permettre à des agents logiciels d'aider efficacement différents types d'utilisateurs dans leur accès aux ressources sur le Web (sources d'information et services). Différent langages de niveau de complexité croissante sont proposés afin de mieux exploiter, combiner et raisonner sur les contenus de ces ressources.

XSD [*XML Schemas Definition*]

XSD est le langage des XML Schema Definition. xsd est aussi l'extension d'un fichier XML Schema.

Voir aussi : Schémas XML.

XSL [*eXtensible Stylesheet Language*]

Le langage XSL permet de présenter visuellement des éléments définis dans un document XML contrairement à XML qui définit plutôt la sémantique des données. Il se divise en deux parties principales, le formatage : application de règles de style sur des éléments XML. Et la transformation : substitution d'un marquage XML en un balisage HTML ou un autre marquage XML.

Référence : http://www.w3.org/TR/xsl/

Xylème [Xyleme]

Xylème est une action de recherche plus récente initiée par l'Inria et qui vise à construire un entrepôt de données de l'ensemble des documents XML du Web. Xylème s'appuie sur l'hypothèse que XML deviendra le langage de description des données du Web de demain.

Voir aussi chapitre 2.4

G.2. Bases de données

Bases de données fédérées [*federated databases*]

Une base de données fédérée est une base de données répartie hétérogène constitue de données fédérées, nécessite donc une architecture qui permet la communication entre les différentes sources de données.

Bases de données hétérogènes [*heterogeneous databases*]

Plusieurs bases de données hétérogènes capables d'interopérer via une vue commune (modèle commun).

Classe [*class*]

On appelle classe la structure interne d'un objet, c'est-à-dire les données qu'il regroupe, les actions qu'il est capable d'assurer sur ses données. Une classe est composée de deux parties :

- les attributs, parfois appelés données membres, il s'agit des données représentant l'état de l'objet.
- les méthodes, parfois appelées fonctions membres, il s'agit des opérations applicables aux objets.

Voir aussi : Objet.

Clé primaire [*primary key*]

Chaque table doit contenir une clé primaire. En effet, les clés primaire doivent être des champs contenant une valeur unique. Elle joue le rôle d'identificateur d'enregistrement (ligne). Les clés influent fortement sur les performances, mais elles sont également nécessaires pour garantir l'intégralité des données.

Clé étrangère [*foreign key*]

Les clés étrangères sont des clés prises dans une table différente. Elles permettent de créer les relations entre différentes tables. Donc la clé étrangère référence la clé primaire d'une autre table afin d'indiquer leur relation.
Voir aussi : clé primaire.

Colonne [*Column*]

Les colonnes d'une base de données correspondent exactement aux attributs du modèle logique. Chaque attribut de ce dernier est représenté par une colonne dans le modèle physique. Un type de données est affecté à chaque colonne créée.

Index [*index*]

Dans les bases de données relationnelles, les index accélèrent l'accès aux informations. La création d'un index dans une colonne ou un groupe de colonnes en garantit le tri le plus rapide possible, au prix de l'utilisation d'un espace disque supplémentaire. Théoriquement, chaque clé étrangère devrait être associée à un index.
Voir aussi : clé étrangère.

NULL [*une valeur à part*]

Dans les bases de données il existe des valeurs particulière qui s'appelle NULL. NULL signifie « indéterminé ». Attention, cela ne signifie pas « vide », c'est complètement différent. La valeur NULL, n'a pas de type. Elle ne peut pas être impliquées dans la moindre opération mathématique puisque sa valeur est indéterminée. L'opération aura donc un résultat indéterminé. En quelque sorte, NULL est un élément absorbant pour la quasi totalité des opérations.

Objet [*object*]

Un objet est une entité cohérente rassemblant des données et du code travaillant sur ses données. Un objet est caractérisé par des attributs, ce sont des variables stockant des informations d'état de l'objet. Les méthodes caractérisant son comportement (ensemble des actions). L'identité qui permet de le distinguer des autres objets, indépendamment de don état.

Procédure stockée [*stored procedure*]

Dans une base de données relationnelle, les procédures stockées permettent d'encapsuler la fonctionnalité de gestion dans une fonction compilée au préalable. Les procédures stockées sont similaire aux triggers et différent uniquement par le fait qu'elles puissent être exécutées à la demande d'un programmeur plutôt qu'automatiquement, en réponse à une action de la base de données, par exemple à la suite de l'insertion d'une table.
Voir aussi : trigger.

Requête [*query*]

On peut définir une requête, par une question posée à un système d'informations conformément à un model.

SQL [*Structured Query Language*]

SQL est un langage de requêtes de bases de données qui est devenu un standard de l'industrie en 1986. Ce langage permet de poser des questions complexes à une base de données et de la modifier. De nombreuses bases de données le supportent, par exemples MSSQL Server, DB2, Oracle, PostgreSQL.

Schéma de base de données [*Schema of database*]

Organisation ou structure d'une BD, provenant généralement de la modélisation de données. Cette structure est décrite à l'aide d'un vocabulaire contrôlé qui nomme des éléments de données et répertorie les contraintes pouvant s'appliquer (type de données, valeurs légales/illégales, etc.). Les relations entre éléments de données constituent également une part importante d'un schéma.

Table [*table*]

Les tables représentent l'élément essentiel des bases de données relationnelles. Une base de données relationnelle est constitué d'une ou de plusieurs tables destinées à stocker des informations. Une table est constituée de lignes. Chaque ligne est divisée en champs (colonnes), lesquels possèdent un certain type données.

Trigger [*déclencheur*]

Les triggers sont au même titre que les contraintes, un élément fondamental dans la structure des bases de données. Les triggers sont définis pour des tables et associés à un événement comme `INSERT` ou `UPDATE`. Dans les situations réelles, les triggers permettent effectuer automatiquement des opérations, mais aussi de nombreuses fonctions internes de la base de données.

Voir aussi : contrainte

UML [*Unified Modeling Language*]

Créé par Grady Booch et Jim Rumbaugh, UML est un langage graphique de modélisation fournissant des éléments syntaxiques pour la plupart des systèmes logiciels, ce qu'UML désigne sous le terme « artifact » (en français: artefact). UML est une norme ouverte, maintenue par l'OMG.

Vue [*view*]

Une vue est une table virtuelle qui contient des informations provenant d'autres tables. Une vue n'est rien d'autre que le résultat d'une instruction SELECT présentée comme une table virtuelle par le système de bases de données. Les vues permettent de simplifier des instruction SQL.

Annexe

A.1 Introduction

Le but de cette annexe est de présenter en premier lieu, une étude comparative entre les deux modèles de données à savoir DTD et schéma XML. Nous découvrons pourquoi les définitions de type de document (DTD) ont rapidement laissé apparaître des carences dans le domaine des contraintes de structures et de données, et pourquoi, aujourd'hui, les XML Schema constitue donc une pièce essentielle dans la sphère XML.

Dans un deuxième temps, nous présentons un bref tutorial couvrant la spécification XML Schema.

A.2 Apports sémantiques du langage XML Schema Definition (XSD)

L'objectif de XSD est de définir des contraintes sur les classes de documents conformes à un même modèle à la différence des DTD, qui ne définissaient que les relations entre les différents composants d'un document. Dans ce qui suit, nous montrons les possibilités offertes par ce langage :

Syntaxe XML

Le fait que les schémas XML soient en réalité écrits en XML constitue probablement la différence la plus évidente entre les schémas XML et les DTD. Les schémas XML décrit un vocabulaire utilisable pour imposer des contraintes aux documents XML.

Typage de données

A titre de rappel, les DTD sont écrites en Extended Backus Naur Form (EBNF). Quant aux schémas XML, ils se servent de XML pour décrire un document, ce qui permet d'éviter aux utilisateurs l'apprentissage d'une autre syntaxe. A la différence des DTD, les schémas offrent à peu prés tous les types de données.

Modèles de contenu

Les modèles de contenu des DTD sont faibles car ils nous permettent seulement de contraindre le document à une liste ordonnée ou de choix. Ils peuvent uniquement contenir zéro, une ou plusieurs occurrences d'un élément. De plus, les éléments nommés ou les groupes d'attributs sont absents. Les schémas, quant à eux, offrent des modèles de contenu plus détaillés et plus solides. Ils n'excluent pas un contenu mixte. En outre, ils peuvent servir à spécifier un nombre exact d'occurrences et à nommer un groupe d'éléments.

Extensibilité

Les caractéristiques telles que les types de données dérivés par l'utilisateur permettant de définir des structures complexes fournissant des moyens évolutifs d'étendre des schémas. La capacité de référencer plusieurs schémas à partir d'un seul document et l'aptitude à réutiliser des parties d'un schéma dans un autre (en particulier, leurs type de contenu) permettent également une bien meilleure gestion de vocabulaires partagés et standard.

Autodocumentation

A la différence des DTD, les schémas XML permettant d'utiliser des élément nommés `annotation` pour commenter le code. Etant donné que les schémas sont écrits en XML, il suffit de relier une feuille de style à un schéma correctement commenté et créer une documentation à cet effet.

Le tableau suivant présente d'une manière simple et claire chaque brique des deux modèles, afin de voir de plus prés leurs différences.

		DTD	***XML Schema***	***Description***
Nœud root		!DOCTYPE	xsd:schema	Élément racine
Eléments		!ELEMENT	xsd:element	Déclaration d'un élément
		PCDATA	xsd:string	Données littérales
		?	minOccurs="0" maxOccurs="1"	Zéro ou une fois
		+	minOccurs="1" maxOccurs="1"	Une seule fois
		*	minOccurs="0" maxOccurs="unbounded"	Zéro ou plusieurs fois
		cardinalité impossible	minOccurs="-5" maxOccurs="+5"	Intervalle [-5, +5]
		type inexistant	xsd :binary	Type primitifs (données binaire)
		type inexistant	xsd :Boolean	Type primitifs (false/true)
		type inexistant	xsd :byte	Type dérivé ([-128, 127])
		type inexistant	xsd:date	Type dérivé (date du jour)
		type inexistant	xsd :time	Type dérivé (heure)
		type inexistant	xsd :uriReference	Type primitif (URI)
		type inexistant	xsd:restriction	Contrainte par restriction
		type inexistant	xsd:extension	Contrainte par extension
Attributs		!ATTLIST	xsd :attribute	Déclaration d'un attribut
		CDATA	xsd :string	Données textuelles
		NMTOKEN	NMTOKEN	Nom XML valide
		NMTOKENS	NMTOKENS	Plusieurs noms XML valide
		(value_1 | value_2 | ….| value_N)	xsd:choice xsd:enumeration	Liste de valeurs possibles, l'attribut ne peut prendre que l'une des valeurs listées.
		ID	ID	Identificateur unique
		IDREF	IDREF	Expression de relation entre deux attributs.
		ENTITY	ENTITY	Référencement des données binaires externes.
		#REQUIRED	use="required"	L'attribut doit toujours être présent.
Commentaires		<!- - texte du commentaire - ->	<!- - texte du commentaire - -> xsd :documentation	Déclaration d'un commentaire
		n'existe pas	Xsd :annotation	

A.3 Spécification du langage XSD

Cette partie s'intéresse à la spécification XML Schema du consortium W3C, trois parties la composent : l'introduction présentant les thèmes clés des schémas et deux autres références respectivement sur les **structures** et les **types de données**.

- Les structures, elles peuvent être élaborées à partir des types de données et utilisées pour décrire un élément, un attribut ou une structure de validation d'un type de document.

- Les types de données, à l'origine des composants plus complexes d'un schéma, constituent les bases fondamentales de XML Schema.

A.3.1 Structure

Élément	*Définition*	*Exemple*
schema	Un élément `schema` commence par l'ouverture d'une élément schéma destiné à accueillir la définition des composants d'un document XML. Il encadre une définition de schéma en se composant comme un élément racine.	**<xsd:schema** ***xmlns:xsd="http://www.w3.org/2001/XMLSchema.xsd"*** xmlns=http://www.site.com/schemaxml/> ... **</xsd:schema>**
element	Permet de représenter un élément XML dans une définition de schéma. L'attribut `name` de `element` contient le nom de l'élément.	**<xsd:element** name="element_enfant" type="xsd:string"**/>**
attribute	permet de représenter un attribut XML dans une définition de schéma. Il contient un attribut `name` qui porte le nom de l'attribut.	**<xsd:attribute** name="attribut_enfant" type="xsd:string" use="required"**/>**
complexType	définit un type de données complexe pour des éléments XML. Il ne peut être inclus que dans les éléments suivants : `element, redefine, schema.`	**<xsd:complexType** name="element_petit_enfantType"**>**
all	Il permet de spécifier, dans un type de données complexe, un à plusieurs éléments devant apparaître une fois ou pas du tout et dans un ordre quelconque. L'élément `all` ne peut être inclus que dans les éléments suivants : `complexType, group.`	**<xsd:all** minOccurs="0" maxOccurs="1"> <xsd:element name="element_1" type="xsd:integer"/> <xsd:element name="element_2" type="xsd:integer"/> **</xsd:all>**
choice	L'élément `choice` propose une structure de choix entre plusieurs éléments possible. Il ne peut être inclus que dans les éléments suivants : `choice, complexType, group, sequence.`	**<xsd:choice>** <xsd:element ref="element_choix_1"/> <xsd:element ref="element_choix_2"/> **</xsd:choice>**
sequence	définit, dans un type de données complexe, un à plusieurs éléments devant obligatoirement apparaître dans un ordre prédéfini. Il ne peut être inclus que dans les éléments suivants : `choice, complexType, group, sequence.`	**<xsd:sequence>** <xsd:element ref="element_1"/> <xsd:element ref="element_2"/> <xsd:element ref="element_3"/> **</xsd:sequence>**
extension	L'élément `extension` propose d'étendre la définition d'un élément ou d'un attribut XML à un autre type de données spécifié. Il ne peut être inclus que dans les éléments suivants : `complexContent, simpleContent.`	**<xsd:extension** base="niveau_don"> <xsd:attribute name="monnaie" type="xsd:string" use="required"/> **</xsd:extension>**
restriction	L'élément `restriction` permet de restreindre les données permises dans un	**<xsd:restriction** base="xsd:nonNegativeInteger"> <xsd:minExclusive value="50"/>

	restreindre les données permises dans un élément ou un attribut XML. Il ne peut être inclus que dans les éléments suivants : `complexContent`, `simpleContent`, `simpleType`.	<xsd:maxExclusive value="100"/> **</xsd:restriction>**
simpleType	Définit un type de données simple pour des éléments XML. L'élément `simpleType` ne peut être inclus que dans les éléments suivants : `attribute`, `element`, `list`, `redefine`, `restriction`, `schema`, `union`	**<xsd:simpleType** name="liste"> <xsd:list itemType="xsd:integer"/> **</xsd:simpleType>**
enumeration	Permet de contraindre la valeur d'un élément ou d'un attribut à une seule possibilité. L'élément `enumeration` ne peut être inclus que dans l'élément `restriction`.	**<xsd:enumeration** value="Célibataire"**/>** **<xsd:enumeration** value="Concubine"**/>** **<xsd:enumeration** value="Mariée"**/>** **<xsd:enumeration** value="Veuve"**/>**
patern	L'élément `pattern` permet de créer un modèle pour la valeur d'une élément ou d'un attribut XML à partir d'une expression régulière. Il ne peut être inclus que dans l'élément `restriction`.	**<xsd:pattern** value="(.)+@(.)+"**/>**
minInclusive	Permet de définir une valeur minimum inclusive pour l'élément ou l'attribut XML. Il ne peut être inclus que dans l'élément `restriction`.	**<xsd:minInclusive** value="100"**/>**
maxInclusive	Permet de définir une valeur maximum inclusive pour l'élément ou l'attribut XML. Il ne peut être inclus que dans l'élément `restriction` .	**<xsd:maxInclusive** value="200"**/>**
minExclusive	Permet de définir une valeur maximum exclusive pour l'élément ou l'attribut XML. Il ne peut être inclus que dans l'élément `restriction`.	**<xsd:minExclusive** value="100"**/>**
maxExclusive	Permet de définir une valeur maximum pour l'élément ou l'attribut XML. Il ne peut être inclus que dans l'élément `restriction`.	**<xsd:maxExclusive** value="200"**/>**
key	L'élément `key` permet de définir un élément clé dans une structure XML. Il ne peut être inclus que dans l'élément `element`.	**<xsd:key** name="IDCommande"> ... **</xsd:key>**
keyref	L'élément `keyref` permet de créer une référence à une clé existante dans le schéma XML. Il ne peut être inclus que dans l'élément `element`.	**<xsd:keyref** name="IDREFCommande" refer="IDCommande"> ... **</xsd:keyref>**
selector	L'élément `selector` permet de définir une expression `XPath` chargée de sélectionner un élément XML afin de lui appliquer une clé. Il ne peut être inclus que dans les éléments suivants : `key`, `keyref`, `unique`.	**<xsd:selector** xpath=".//produit"**/>**
field	L'élément `field` permet de sélectionner par l'intermédiaire d'une expression `XPath` , des éléments ou des attributs destinés à être utilisés comme clé. Il ne peut être inclus que dans les éléments suivants : `key`, `keyref`, `unique`.	**<xsd:field** xpath="./@idCom"**/>**
annotation	L'élément `annotation` propose une structure permettant de fournir des informations applicatives ou destinées à l'utilisateur dans un schéma.	**<xsd:annotation>** Liste des codes de services établie le 12/04/03 **</xsd:annotation>**
documentation	spécifie une information à propos du schéma destinée à l'utilisateur. L'élément `documentation` ne peut être inclus que dans l'élément `annotation`.	**<xsd:documentation>** nom de la base de données est linc **</xsd:documentation>**

A.3.2 Types de données

Nous avons déjà vu que les types de données présentaient des avantages et que les schémas XML fournissaient deux sortes de types de données principaux :

- **Les types de données de base**, non définies par d'autres types de données ;
- **Les types de données dérivés**, définis par les types existants.

A.3.2.1 Types de données de base

Les types de données primitifs constituent la base des autres types et sont eux-mêmes indéfinissables par des composants moindres. Par définition, les types primitifs ne présentent ni contenu d'élément ni attribut, puisqu'ils forment les types de base à partir desquels tous les autres types sont dérivés [WIL01].

Le tableau ci-dessous présente ces types primitifs et leurs description.

Type primitif	Description
`string`	Toute chaîne de caractères XML légale.
`boolean`	« true » (vrai) ou « false » (faux)
`float`	Concept courant de nombres réels correspondant à des nombres à virgule flottante de 32 bits de données en simple précision.
`double`	Concept courant de nombres réels correspondant à des nombres à virgule flottante de 64 bits de données en simple précision.
`decimal`	Nombre décimal en précision arbitraire.
`timeDuration`	Durée de temps.
`recurringDuration`	Laps de temps répétant à une fréquence déterminée à partir d'un moment précis.
`binary`	Données binaires arbitraires.
`uriReference`	URI.
`ID`	Type d'attribut `ID` conformément à la recommandation XML1.0.
`IDREF`	Type d'attribut `IDRF` conformément à la recommandation XML1.0.
`ENTITY`	Type d'attribut `ENTITY` conformément à la recommandation XML1.0.
`Qname`	Nom qualifié conformément à la recommandation XML1.0.

A.3.2.2 Types de données dérivés

Ici, un type de données est défini selon un type de données existant (type de base). Il peut comprendre des attributs et un contenu d'élément ou les deux.

Type primitif	Description	Type de base
CDATA	Chaîne de caractère pouvant comporter des espaces.	String
token	Chaîne marquée par un jeton.	CDATA
NMTOKEN	Type d'attribut NMTOKEN de XML1.0.	Token
NMTOKENS	Type d'attribut NMTOKENS de XML1.0.	NMTOKEN
Name	Représente un nom XML.	Token
integer	Représente un nombre entier.	Integer
Int	Représente un nombre entier long dont l'intervalle est {-2147483648, 2147483647}	Integer
long	Dérivé de integer, la valeur de maxInclusive étant fixée à 9223372036854775807 et celle de minInclusive à –9223372036854775808.	Integer
short	Dérivé de int, la valeur de maxInclusive étant fixée à 32767 et celle de minInclusive à –32768.	Int dérivant de long
byte	Dérivé de short, la valeur de maxInclusive étant fixée à 127 et celle de minInclusive à –128.	Short
date	Jour particulier .	TimePeriod de recurringDuration
nonPositiveInteger	Nombre entier négatif incluant le zéro.	Integer
negativeInteger	Nombre entier négatif dont la valeur maximale est –1.	Integer
nonNegativeInteger	Nombre entier positif incluant le zéro.	Integer
positiveInteger	Nombre entier positif dont la valeur minimale est +1.	integer

A.4 Conclusion

Dans cette annexe, nous avons constaté que les schémas XML offre une autre solution bien plus puissante et descriptive que les DTD. Nous avons découvert également la spécification XML Schema telle qu'elle est défini par le W3C (structure et type de données).