MASTER EN SCIENCES INFORMATIQUES

Etude d'un langage visuel d'interrogation de base de données

Duchene, Lilian

Award date:
2004

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année Académique 2003 – 2004

Etude d'un langage visuel d'interrogation de Base de Données

Lilian DUCHENE

Mémoire présenté en vue de l'obtention du grade de

Licencié en Informatique.

TABLE DES MATIERES

0 .Abstract

Nous savons qu'une base de données tient de la théorie des ensembles et qu'ils représentent une partie du domaine d'expertise d'une entreprise, d'un utilisateur...

Mais quels sont les moyens aujourd'hui pour récupérer ces données ?

Le langage SQL, des formulaires réalisés à la demande (suivant la charge d'un service informatique) pourraient être des solutions. Tant d'effort, d'une part et d'autre, qui coûtent cher.

N'y en a-t-il pas d'autres ?

Ce mémoire va présenter le problème des interrogations des bases de données et va présenter une solution tirée de la théorie des Visual Query Language (VQL) Ces langages Visuels d'interrogation de bases de données vous seront présentés au chapitre « Etat de l'art » et une solution idéale ensuite en découlera. La solution réduite aura pour but de réaliser un éditeur graphique qui démontrera la faisabilité de la solution idéale. Un exemple à la fin de ce chapitre permettra de mieux comprendre son fonctionnement. Pour pouvoir interpréter le VQL et le traduire en un langage d'interrogation de base de données (SQL par exemple) l'éditeur s'appuie sur un arbre syntaxique qu'il transmettra à un compilateur. Ce point est traité dans le chapitre 'Solution réduite'.

Mots clés : hypergraphe des relations spatiales, modèle d'hypergraphe, base de données, langage, visuel, interrogation, requête

---

We know that a database come from the theory of sets and that they represent a part of the field of expertise of a company, of a user...

But which are the means to recover these data today?

Language SQL, the forms carried out with the request (according to the load of a Information Systems department) could be solutions. Such an amount of effort, on the one hand and the other, which are expensive.

Are of them there no others?

This memory will present the problem of the queries of the databases and will present a solution drawn from the theory of Visual Query Language (VQL) These Visuals Query Language of databases will be presented to you at the chapter "Etat de l'art" and an ideal solution then will result from this. The purpose of the reduced solution will be to carry out a graphic editor who will prove the feasibility of the ideal solution. An example at the end of this chapter will make it possible to better include/understand its operation. To be able to interpret the VQL and to translate it into a language of basic interrogation of data (SQL for example) the editor is pressed on a syntactic tree which it will transmit to a compiler. This item is discussed in the chapter ` Solution réduite'

Key words : spatial relationship hypergrah, hypergraph model, database, language, visual, query

Remerciements:

Je désire remercier le Professeur Vincent Englebert pour sa patience et sa disponibilité lors de l'élaboration de ce mémoire ainsi que la connaissance de qualité qu'il m'a transmise.

Un merci tout particulier à Christine pour sa compréhension et son soutien, à mon beau-père pour ses conseils avisés et son encouragement.

Merci à mes parents et mes amis qui m'ont soutenu et supporté pendant les moments les plus difficiles.

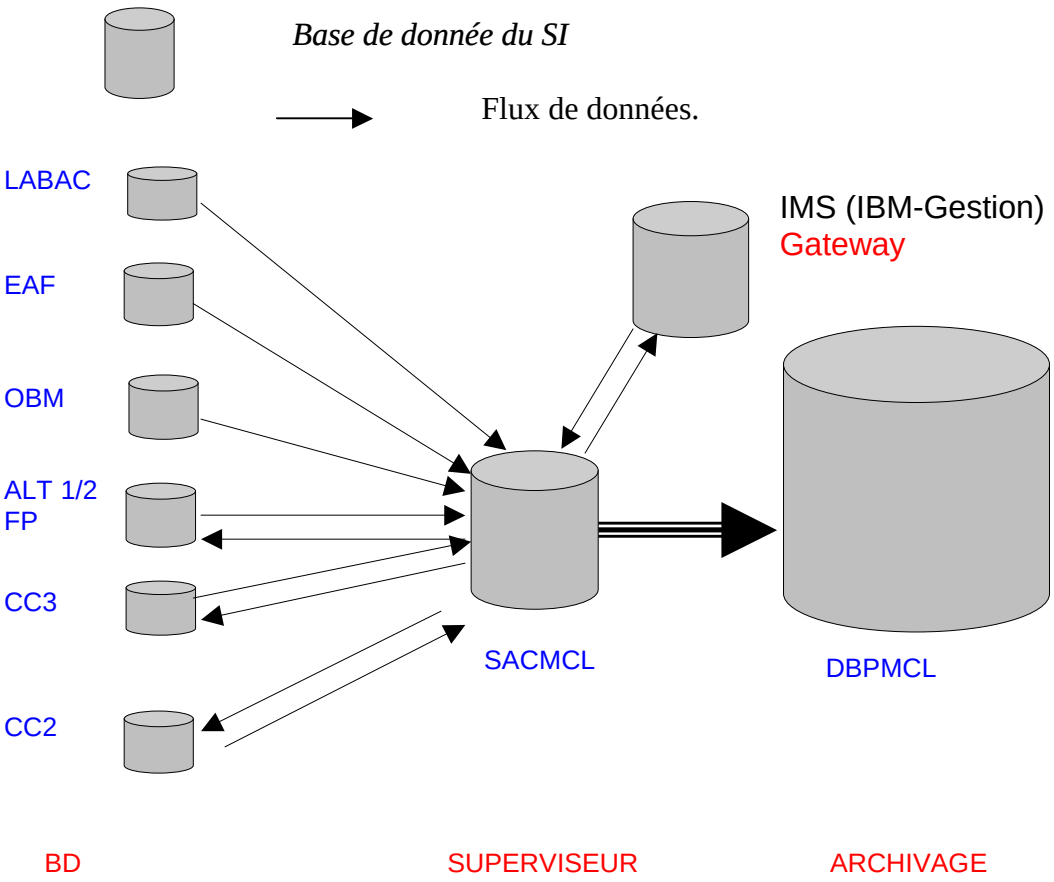
1 .Introduction

CARSID[1] (CAROLO-SIDERURGIE, anciennement COCKERILL-SAMBRE) produit des brames d'acier de différentes dimensions, de différentes caractéristiques et qualités, et ceci en production continue.

Son système d'information[2] a comme responsabilité de contrôler et de gérer les flux de données produites par le processus industriel[3], mais aussi de les stocker. La quantité de données mémorisées au bout d'une journée, est tellement importante, que les garder augmenterait le temps de réponse vers le processus industriel. D'où la présence dans ce SI d'une base de données *ARCHIVAGE*. Son rôle est de garder l'historique des valeurs principales, et de permettre aux utilisateurs de consulter les données sans perturber la production.

Schéma de l'architecture du SI :

Légende :



[1] Situé à Marcinelle (Charleroi)
[2] Appelé SI dans la suite du document.
[3] Ceci à travers un contrôle-commande, appelé niveau 2-3 chez COCKERILL-SAMBRE.

Fonctions des serveurs (bases de données) :

- **EAFMCL, CC2/3MCL :**

- Respectivement liés aux ateliers EAF, OBM, CC 's
- Leurs rôles sont de récolter, fournir les données aux acteurs de l'exploitation (supervisions, automates, …), et de les contrôler.

- **SACMCL**
  - Il récolte les informations venant des différents ateliers
  - Permet d'effectuer les transferts de données suivants :
    - De EAF[4], OBM[5] vers FP[6],ALT 's[7]
    - De FP, ALT's vers CC's[8]
    - De CC 's vers Archivage des informations
  - Chaque transfert ne peut s 'effectuer que par le biais de SACMCL

- **LABAC :**

- Situé au laboratoire de l 'aciérie, il est chargé de récolter les informations des analyses des échantillons d 'acier, prélevés sur les différents ateliers

- **DBPMCL :**
  - Il est chargé de l'archivage des données
  - Il est la seule référence sur l 'intégrité des informations

Pour consulter les bases de données sur ces serveurs et pour respecter les flux de données indiqués sur le schéma de la page précédente, l'équipe en charge du système d'informations maintient des applications.

La base de données ARCHIVAGE est en constante évolution puisqu'elle subit tous les changements des autres bases de données. La maintenir est difficile, mais ceci est valable aussi pour les applications qui la consultent.

Les utilisateurs passent par des applications pour la consulter, car leur métier est d'analyser, optimiser, maintenir le processus industriel et non de connaître le langage de consultation (SQL[9][10]) et la structure des tables !

Ce qui signifie qu'à chaque nouvelle transaction sur la base de données ARCHIVAGE, un utilisateur doit demander à l'équipe SI de réaliser ou modifier une application.

Nous constatons lors d'un audit qu'il existe beaucoup de maintenances sur la base de données en cas d'évolution du SI, beaucoup de petites applications de consultation, peu de standardisation. La réalité permet d'ajouter que la base de données ARCHIVAGE est un nœud sur l'architecture du SI donc une zone sensible et on constate beaucoup trop de connexions dues aux applications de consultation. La base de données est réalisée sur un modèle conceptuel basé sur le fonctionnement des ateliers donc il y a peu ou pas du tout de vue globale sur l'entièreté du processus.

Ce mémoire va présenter la problématique de la consultation de la BD ARCHIVAGE pour les utilisateurs néophyte dans ce domaine. Puis il tentera de proposer une solution en vue d'améliorer cet accès aux informations. Pour cela la

[4] Four électrique : transforme la ferraille en acier
[5] Convertisseurs : transforme la fonte en acier
[6] Four poche : change les caractéristiques de l'acier provenant du four électrique
[7] ALT : change les caractéristiques de l'acier provenant de l'OBM
[8] Coulées Continues : durcit l'acier et le découpe en brames.
[9] SELECT QUERY LANGAGE
[10] Toutes les bases de données du SI sont des bases de données relationnelle.

structure du mémoire respectera le schéma suivant : problématique, état de l'art, solution idéale, solution réduite, et il montrera la consistance des solutions en donnant un exemple.

2 .Problématique

Un système d'information est au cœur d'une société dans les échanges, le traitement et le stockage des données. Dans le cadre de notre de travail nous verrons un système d'information comme un ensemble de bases de données. La plupart sont constituées de systèmes de persistance de type opérationnel ET de type archivage. Ces derniers ont comme rôle de garder l'historique des valeurs principales du processus industriel, et de permettre aux utilisateurs de les consulter sans perturber la production. La structure de ce type de base de données est en constante évolution car étant dépendante de la structure de celles de type opérationnel, elles en subissent tous leurs changements. Les maintenir en devient difficile, mais ceci est valable aussi pour les applications qui les consultent. Les utilisateurs sont obligés de passer par celles-ci. Leur métier est d'analyser, optimiser, maintenir le processus industriel et non de connaître des langages de consultation (SQL) ou la structure des tables d'une base de données.

Les besoins des utilisateurs sont de récupérer des données qu'ils pourront observer, analyser, traiter puis mettre en forme. Pour cela il leur faut un moyen pour pouvoir les récupérer. La plupart des SI fournissent 2 moyens d'interrogation aux bases de données d'archivage :

- des applications spécifiques développées en interne.
- des produits capables d'exécuter des requêtes SQL, tels que Microsoft ACCESS ou Microsoft QUERY.

En résumé un utilisateur, pour consulter la base de données ARCHIVAGE, doit demander une application (si elle n'existe déjà) à l'équipe du SI pour qu'elle envoie une transaction[11] et mette en forme le résultat. Ou alors, il doit apprendre le langage SQL.

Demander un développement en interne a quelques inconvénients. Tout d'abord l'équipe du SI doit s'organiser en projet, c'est-à-dire : prévoir le budget, les moyens en personnel et matériel requis, effectuer des interviews pour connaître d'autres éventuels utilisateurs, analyser les besoins puis le développement, programmer l'application, et enfin la mettre à disposition des utilisateurs. Le temps de développement n'est pas du tout négligeable. Ceci serait acceptable si l'utilisateur pouvait anticiper sa demande. La réalité est tout autre. A l'heure actuelle, le rendement est primordial et chaque étude a pour but de toujours l'améliorer (de diminuer les coûts et d'augmenter la production) Les utilisateurs qui sont la plupart des ingénieurs ou des responsables chef de zone sont donc constamment sous la coupe de la Direction qui demande des chiffres et encore des chiffres. De ce fait, il est pratiquement toujours impossible d'anticiper la demande de nouvelles transactions, tout en sachant que l'application ne répondra qu'aux requêtes demandées et seulement celles-ci.

Le SI a donc prévu une alternative : *les produits capables d'utiliser les requêtes SQL*. Seulement ceci ne peut être utilisé que de 2 manières : soit l'utilisateur doit apprendre le SQL, soit le SI lui fournit la requête. Dans les 2 cas, le SI a gagné en temps de réponse aux besoins de l'utilisateur. Mais un inconvénient non négligeable apparaît : la diffusion de l'information de la structure de la base de données et l'accès aux données. Dans le cas de l'application, on peut contrôler par le biais d'un usermanagement[12] les accès aux données ou à un ensemble de données. Mais dans le cas des requêtes SQL, la granularité du contrôle peut descendre au maximum au niveau de la table

[11] Question posée à une base de données.
[12] Gestion des utilisateurs et de leurs autorisations d'accès aux informations d'un site

(impossible de réaliser un accès sur base d'un rôle[13] par exemple) Ceci est pour la gestion des accès, mais la diffusion de l'information et sa maintenance est tout aussi importantes.

La base de données ARCHIVAGE subit beaucoup de changements, puisqu'une modification d'une base de données OPERATIONNELLE oblige une modification de la base de données ARCHIVAGE. Donc chacune de ces modifications perturbe ou peut perturber une requête d'un utilisateur. La maintenance de celles-ci nécessite donc un inventaire (qui a eu quoi) qui doit permettre la mise à jour du SQL envoyé. Ceci est valable dans le cas où le SI a fourni le SQL, mais dans le cas où l'utilisateur produit lui même le sien, le SI doit lui documenter la modification de la base de données. Cette dernière solution réduit effectivement le temps de réponse aux utilisateurs, mais augmente la charge en maintenance de l'équipe SI. Cette solution donne dans la réalité une charge de 80% pour la maintenance et 20% dans le développement sur certains membres de l'équipe SI.

Ces 2 moyens de communication proposés par le SI donnent comme inconvénients que d'un côté l'utilisateur perd du temps, et de l'autre c'est le SI qui en perd et est donc réticent à l'employer. Pourtant vu de l'extérieur quelle que soit la solution tout semble fonctionner. Une étude a permis de savoir pourquoi, la réponse est : l'existence d'un 2nd SI.. Ce système est basé sur l'échange de fichiers Excel envoyés par mail. L'ensemble des fichiers forme une base de données. L'explication de son existence en est simple : le besoin important des utilisateurs à obtenir l'information dans les temps. Ceux connaissant le SQL réalisent des requêtes dans des fichiers EXCEL et les mettent en forme grâce aux macros.

Il n'y a pas de bonne ou de mauvaise solution, chacune essaie de répondre au mieux à certains des besoins, mais n'y a-t-il pas d'autres solutions ?

Si on reprend les différents besoins des différentes parties. On arrive à cet état des lieux :

Les utilisateurs :

- Sont ingénieurs ou CRZ (Chef Responsable de Zone).
- ® Aimeraient extraire des données.
- Ne connaissent pas la BD
- Peu de connaissance en SQL.
- Leurs transactions les plus compliquées regroupent des données c'est-à-dire du type jointure uniquement

® Réalisent des rapports, des statistiques avec les données de la BD Archivage.

L'équipe SI :

- Veut diminuer la maintenance
- Veut diminuer la difficulté des applications
- Veut ne pas augmenter la complexité du système

[13] par exemple : Modèle des autorisations d'accès utilisé pour le portail de la Sécurité Sociale, basé sur les rôles Citoyens, Fonctionnaires, Entreprise, Médecin, Mandataire, … cfr http://www.belgium.be/eportal/application?origin=searchResults.jsp&event=bea.portal.framework.internal.refresh&pageid=contentPage&docId=30716

En résumé, la nouvelle solution, développée par le SI, devra :

Améliorer la communication entre l'homme et la machine. C'est-à-dire que l'utilisateur interroge la base de données avec ses propres requêtes librement sans dépendance à une application, sans connaître la structure de la base de données et avec ses propres mots.

3 .Etat de l’art

(Théorie) Les différentes catégories et sous-catégories de VQS

Ce chapitre se base sur l'article « Visual Query Systems for Databases : A survey » [CATARCI 95] écrit par CATARCI, COSTABILE, LEVIALDI ET BATINI. Cet article permet de connaître les différents systèmes à interface graphique qui interrogent des bases de données et ayant des utilisateurs sans connaissance sur les méthodes d'interrogation des SGBD[14].

L'article approche dans un premier temps la problématique des utilisateurs face à une base de données. D'où l'étude et la réalisation des « Visual Query Systems for Database »[15] qui permettent d'améliorer l'efficacité de la communication entre l'homme et la machine. Ainsi l'article présente l'objectif le plus important de ces outils, qui est de déterminer la nature du dialogue homme-machine, afin de pouvoir les comparer. Pour cela ces systèmes se basent sur un langage, appelé « Visual Query Language[16] », qui détermine la syntaxe de la représentation visuelle des requêtes créées dans le but d'interroger des bases de données.

En extrayant de cet article les différentes catégories de langages, puis en les confrontant à l'analyse des besoins des utilisateurs, nous pensons obtenir le VQL (ceci sera montré au chapitre suivant : ***Solution idéale***) qui répondra au mieux à la problématique du chapitre précédent.

[14] Systèmes de gestion de Base de données
[15] appelé VQS par la suite
[16] appelé VQL dans la suite du document

3.1 Introduction

Aujourd'hui, la plupart des moyens de communication entre les utilisateurs et les systèmes informatiques sont basés sur l'utilisation d'outils graphiques du type **WIMP** : Window, Icône, Menu, Pointer.

Le but des outils de type WIMP est de :

- Raccourcir la distance entre le modèle mental de l'utilisateur et la représentation de la réalité proposé par la machine[17] [VESALE 01].
- Réduire la dépendance de l'utilisateur avec le langage naturel.
- Faciliter l'apprentissage des fonctionnalités basiques qui permettent des interactions avec le système d'information.
- Augmenter le taux de performance obtenu par les utilisateurs experts. En partie, grâce à la possibilité de définir de nouvelles fonctions et objectifs.
- Réduire le taux d'erreur.

Pour respecter les buts cités ci-dessus, les VQL doivent prévoir un formalisme sur *la taille, l'intensité, la texture, les formes, l'orientation, la couleur* des composants du langage graphique manipulé par l'utilisateur. Ces mêmes composants doivent être aussi compris et manipuler par la machine. Donc un compromis entre le Visuel[18] et le Formel[19] doit être prévu.

3.2 Classification des critères des VQL (Taxonomies[20] des VQL) :

Pour classifier les VQL, il faut pour cela s'attarder sur la manière dont travaille un utilisateur lorsqu'il effectue une requête et s'attarder sur ses attentes d'un outil.

Un utilisateur qui veut interroger une base de données a déjà délimité son domaine d'action. C'est-à-dire qu'il a déjà réalisé la requête en termes provenant du process sur son domaine d'expertise.

L'objectif demandé à l'utilisateur est de traduire cette requête en termes mis à sa disposition. On appellera l'effort de traduction : *la formulation de la requête*, et l'ensemble des termes mis à sa disposition : *la représentation de la réalité du centre d'intérêt de l'utilisateur*. Ces 2 objectifs ont un critère en commun, c'est celui de la *Représentation Visuelle*.

---

[17] Définition : (extrait de [VESALE 01])
*L'**effort cognitif** sera d'autant plus faible que les **distances sémantique** et **articulatoire** seront réduites.*
*La **distance sémantique** est faible si l'on peut facilement exprimer dans le langage de l'interface ce que l'on veut réaliser, sans faire appel à des notions étrangères et comprendre ensuite le résultat.*
*La **distance articulatoire** est faible si l'on peut déduire facilement de la forme d'une expression sa signification.*

[18] Visuel : généré, compris et communiqué par l'utilisateur.

[19] Formel : analysé, manipulé et maintenu par un ordinateur.

[20] Science des lois de classification.

Après avoir réalisé une première fois la requête, l'utilisateur va la critiquer pour pouvoir l'optimiser. On l'appellera tout naturellement : *optimisation de la requête.*

La suite du chapitre montrera les différentes techniques employées pour répondre à ces deux critères : représentation visuelle et optimisation de la requête.

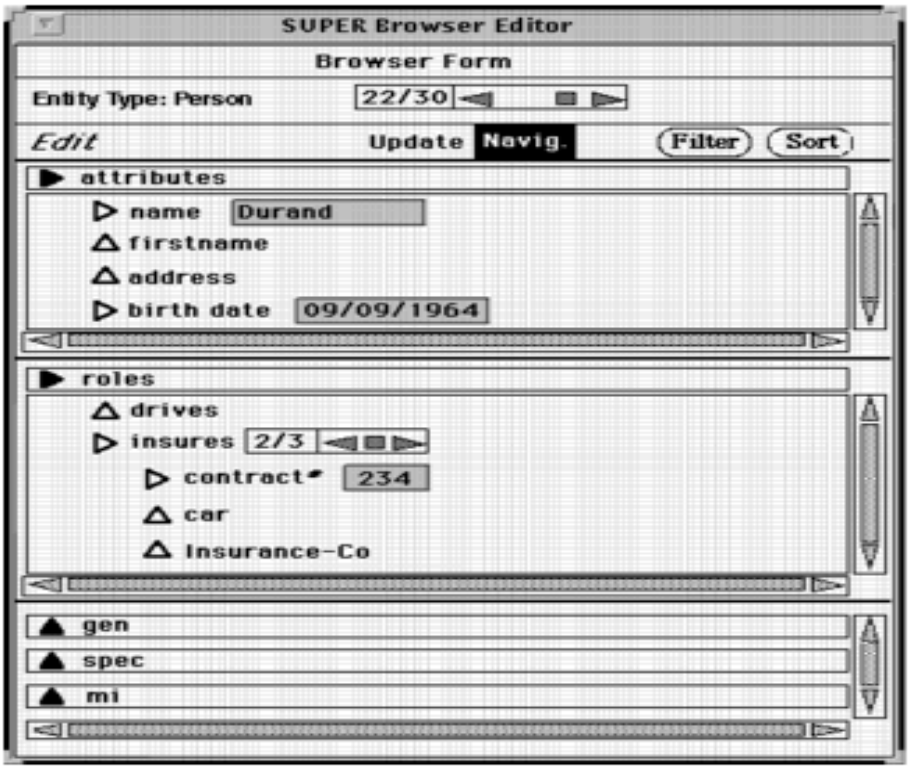
3.2.1 Critère 1 : Représentation Visuelle

Représentations des VQL's
Les outils actuels sont réalisés avec différents choix de représentations :

- Par Formulaire :

Principe : succession de formulaires posant des questions.

Le plus connu est *Query by Example* :



the form browser

- Par Diagramme :

Principe : Le diagramme a pour but de représenter (complètement ou partiellement) le domaine que l'utilisateur veut interroger. Les diagrammes montrent le domaine en reliant les concepts entre eux. La sémantique du langage se représente par les liens entre concepts.

Quelques outils connus:
SUPER [AUDDINO 92],
HYGRAPH,
VQUERY [JONES],
QBD [CATARCI 93]

- Par Icône :

Principe : Les icônes permettent d'ajouter un aspect 3D qui ajoute les notions de distance et de profondeur à la sémantique du langage visuel.

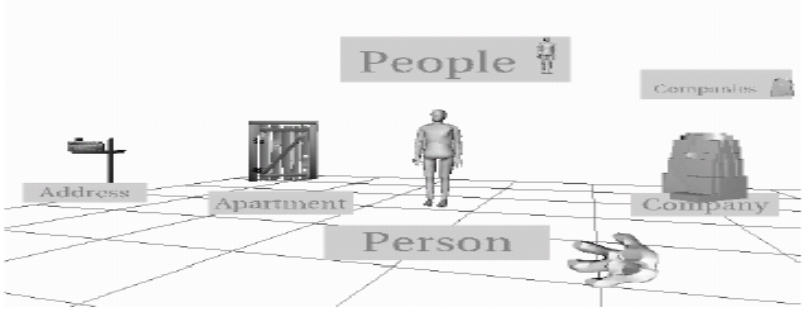


Figure 17: Database Schema

Figure 18: Person attributes

Sample of query in KALEIDOQUERY [KALEI 00].

Quelques outils connus:
QBI [BALKIR 95],
KALEIDOQUERY [KALEI 00]

- Par Assemblage :

Principe : L'assemblage des concepts de la requête va permettre une vue que recouvrera la réponse dans le domaine complet de l'utilisateur.

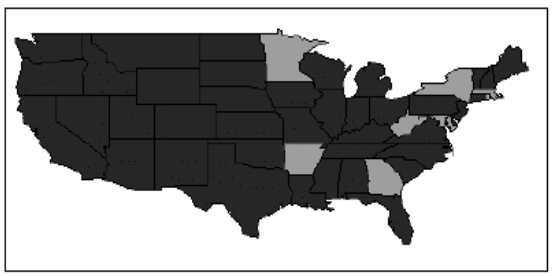


Figure 2. A basic DataSplash visualization.

Figure 3. A visual selection.

Sample of query in VIQING [OLSTON 98].

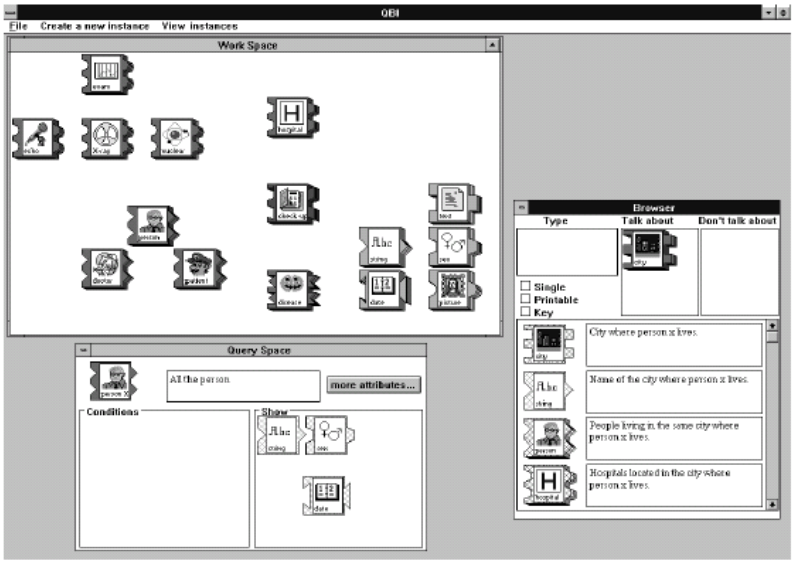


Figure 3. The QBI Interface

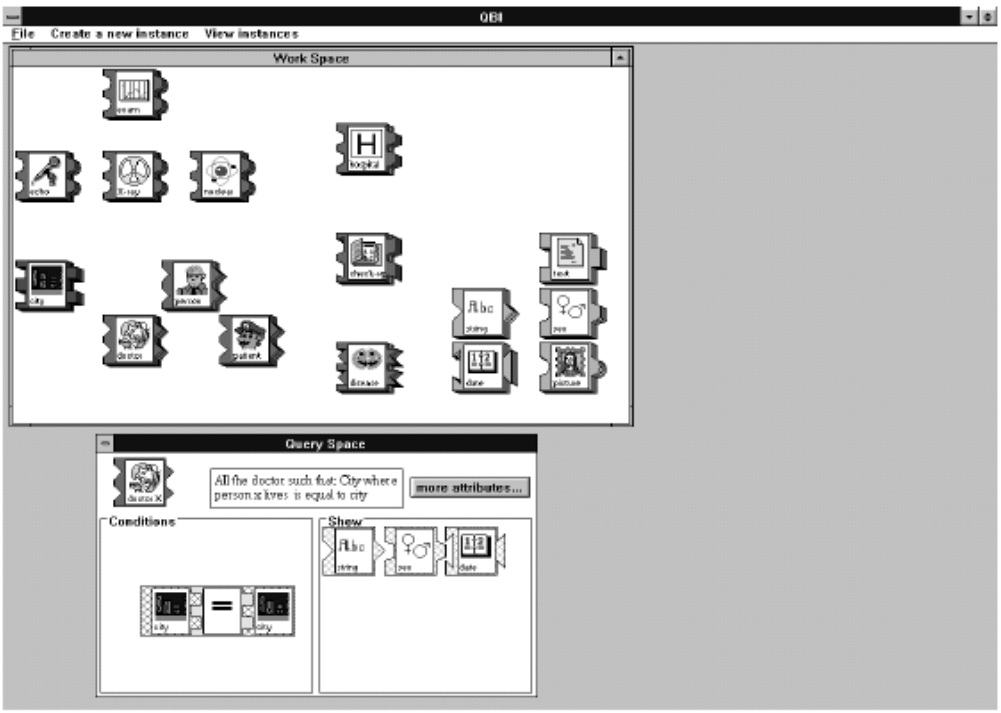


Figure 4. A Query

Quelques outils connus:
VIQING [OLSTON 98],
VISUAL [BALKIR 95],
QBI [BALKIR 95],

- Représentation Spatiale :

Principe : Cette représentation est basée sur la théorie des ensembles. C'est-à-dire que l'utilisateur va découper en plusieurs sections puis il indiquera les actions à effectuer sur ces sections.

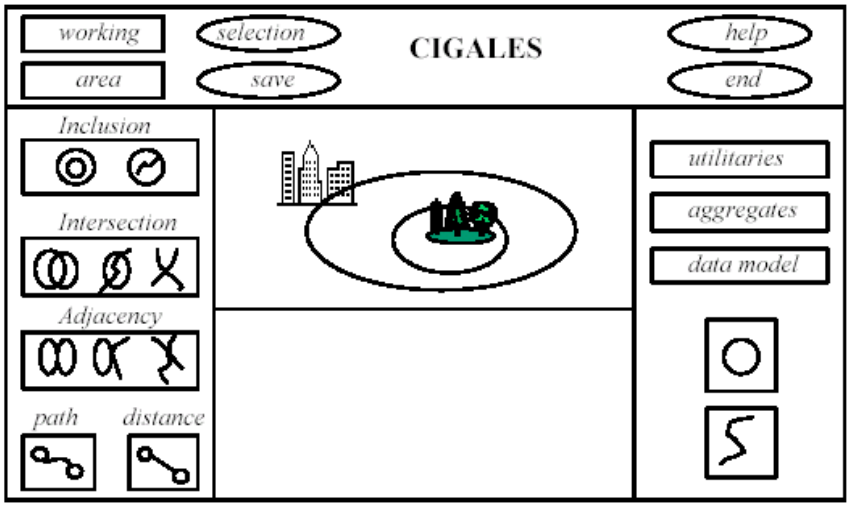


Figure 1. *Le langage visuel Cigales*

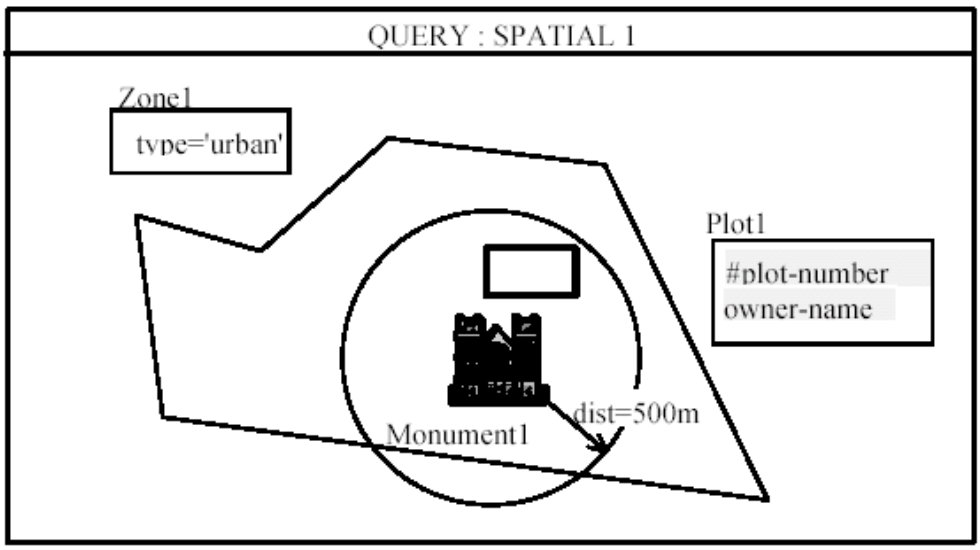


Figure 2. *Le langage visuel Sketch!*

Quelques outils connus :
CIGALES
SKETCH !

- Hybride :

Mélange des 2 techniques ci-dessus, c'est-à-dire :

- § Formulaire et Diagramme,
- § Diagramme et icônes,
- § ou Formulaire - Diagramme et icône

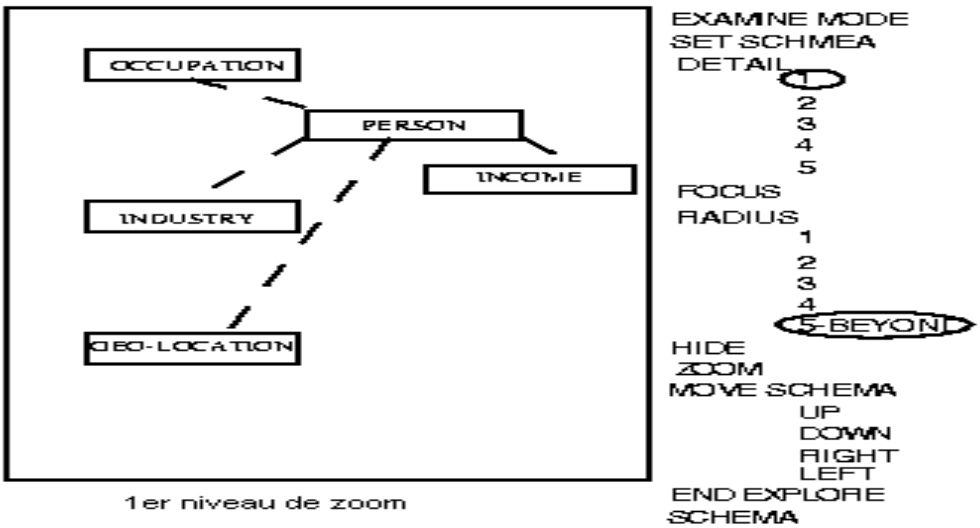
Techniques des VQL's

Dans chacun des outils, on retrouve les différentes techniques permettant de répondre aux 2 objectifs de ce critère : *la formulation de la requête*, et *la représentation de la réalité du centre d'intérêt de l'utilisateur*

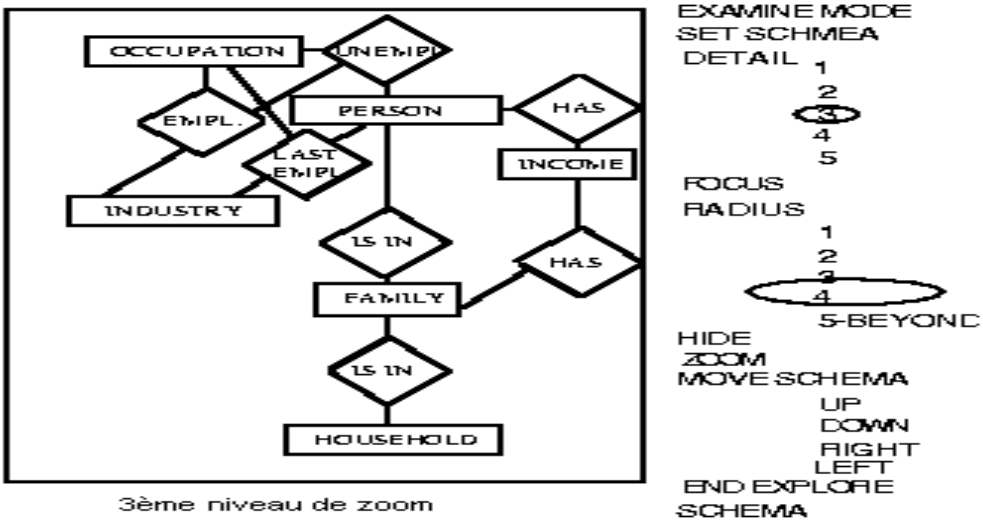
Pour la représentation de la réalité du centre d'intérêt de l'utilisateur, nous avons les techniques du :

- **Top-down :**

Elle consiste en une séquence de raffinements, et de zooms hiérarchiques, dans les différentes présentations de la vue (modèle) proposée à l'utilisateur.



1er niveau de zoom



3ème niveau de zoom

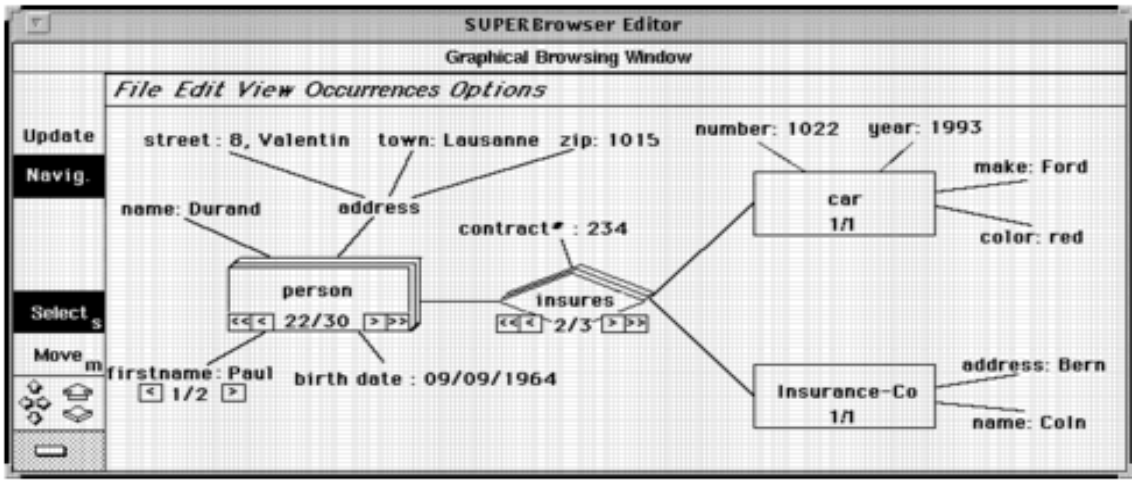
- **Browsing :**

Plusieurs présentations :

§ Intensional :

Elle est exécutée sur le schéma de la base de données, plus précisément sur le schéma conceptuel. L'approche, suivie par exemple dans QBD * correspond à une méta-requête sur la structure du schéma :

a) en demandant tous les chemins existants reliant deux concepts et l'indication des conditions sur la longueur du chemin et/ou la présence des concepts particuliers ;
b) en demandant tous les concepts qui possèdent une ou plusieurs propriétés ;
c) donné un groupe de concepts, choisissant tous leurs concepts voisins.



The Diagrammatic Browser in SUPER.

§ Extensional :

Elle est exécutée sur la prolongation de la base de données. Deux approches différentes peuvent être précisées. La première, présentée par exemple dans G+, consiste en montrant les occurrences de la base de données comme sommets d'un graphique, où les bords représentent des liens existants entre de telles occurrences. La seconde, adoptée par exemple dans SUPER, commence à partir de la présentation intensional de la base de données, puis choisit un objet, et passe en revue ses exemples. Un genre particulier de lecture rapide est fourni par LID (*Living In the Database*) L'utilisateur navigue à travers un simple E-R tuple et voit la base de données en perspective de ce tuple. En commençant une session, l'utilisateur d'abord choisit une entité et puis ensuite choisit, dans une liste de tous les tuples d'entité, le tuple où il veut arriver (appelé le tuple courant). Le système montre alors les relations de ce tuple avec les autres. Quand un tuple d'une entité liée est choisi, il devient le nouveau courant et l'affichage change selon cette nouvelle perspective.

§ Mixte :

Dans ce cas-ci l'activité de lecture peut être exécutée sur les présentations intensional et extensional de la base de données. Par exemple certains interfaces utilisateur surmontent les inconvénients des paradigmes de lecture rapide (par exemple, inefficacité, "short-sighted" ; la navigation) en présentant trois modèles (modèle interne, modèle d'interaction, modèle d'affichage) pour représenter l'information. En particulier, le modèle d'interaction divise chaque voisinage d'objet dans les sous-ensembles homogènes et définit les structures appropriées pour soulager la présentation, et l'interaction avec, des descriptions d'objet.

Pour l'objectif « *Formulation de la requête* », nous avons les techniques :

- **Par Schéma de navigation :**

Trois manières de présenter le schéma existent :

§ Par « connexion arbitraire de chemins » :

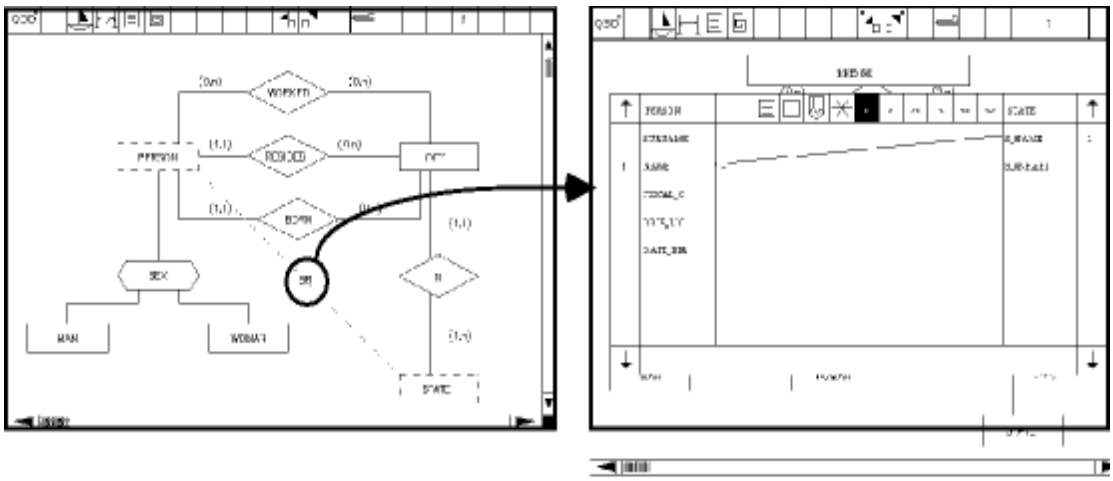
Dans cette approche, l'utilisateur parcourt de manière arbitraire le schéma conceptuel de base de données, afin de choisir les concepts impliqués dans la question. Un exemple est OQL, un langage d'interrogation orienté objet, dans lequel la base de données est représentée au moyen d'un diagramme sémantique. En OQL, une question est exprimée en passant en revue d'abord un diagramme sémantique des classes d'objet et en sélectionnant les objets tout au long du parcours ceux intervenant dans la question.

§ Par « connexion hiérarchique de chemins » :

Dans ce cas-ci l'utilisateur se concentre sur un concept de la base de données et alors le système établit une vue hiérarchique de la base de données avec le concept choisi comme racine. Par exemple, dans GORDAS l'utilisateur choisit d'abord un concept racine qui détermine la direction de la référence pour les relations impliquées. Des attributs de relation sont assignés à l'entité au niveau le plus bas dans la hiérarchie. Puis, l'utilisateur fournit les conditions de choix. D'abord, des conditions sur les attributs de l'entité de racine sont indiquées, et, plus tard, ceux qui impliquent les entités relatives. Il est à noter que différentes entités de racines peuvent être indiquées pour la même question provoquant différentes vues.

§ Par « chemin non connecté » :

Dans les exemples précédents les concepts voisins peuvent être atteints à partir du concept central par des chemins explicitement représentés dans le schéma. Cependant, l'utilisateur peut être intéressé par des nouvelles relations entre concept qu'il devra dans ce cas construire lui-même. Cette approche est typique dans les systèmes relationnels, employant le prétendu opérateur OUTER JOIN. Par exemple, dans HIQUEL l'utilisateur commence par choisir un nom d'entité dans une liste puis le système dessine sur l'écran une table vide pour l'ensemble d'entité, indiquant l'attribut appelé. Point par point, l'utilisateur construit la question, supprimant d'abord les attributs non-appropriés, et écrivant ensuite des conditions aux entités choisies, et ainsi de suite. Les opérateurs outer join peuvent également être exprimés en modèle entité-association. Par exemple, dans QBD * un primitif de question est disponible joindre des entités non explicitement liées dans le schéma.

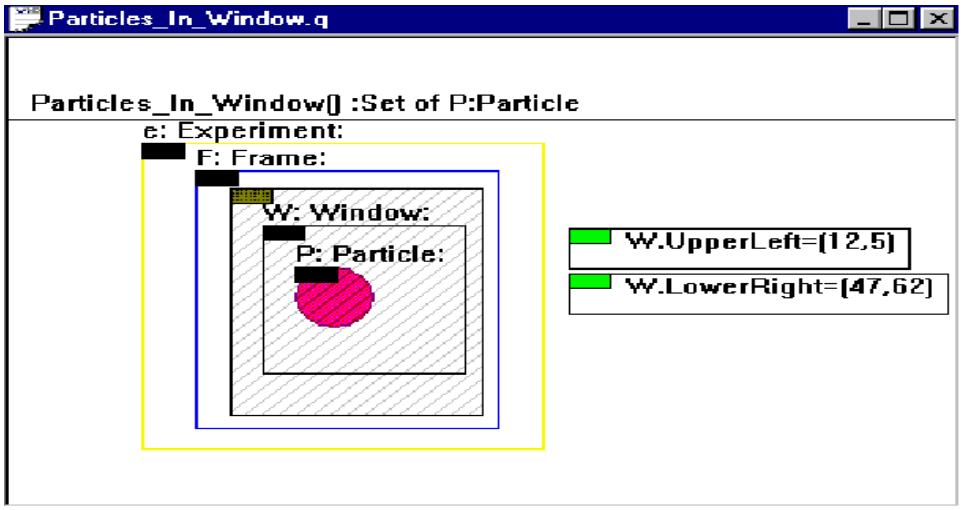


Unconnected Path in QBD*

- **Par Sous-requête :**

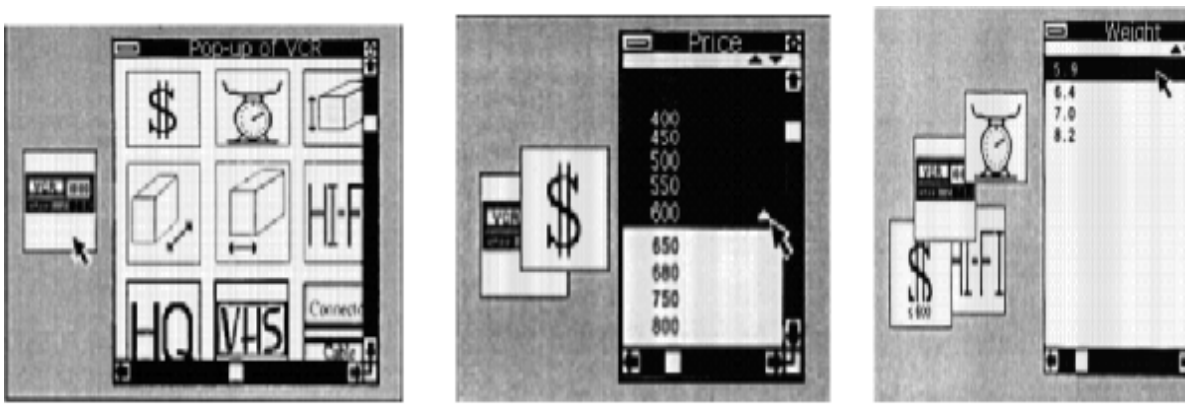
Plusieurs présentations :

§ Par « composition de concept » :
L'utilisateur a la possibilité d'utiliser des concepts (requêtes) déjà prédéfinis. La sortie d'un concept est produit par l'alimentation d'un autre en entrée.



Class - Sub-Class relationship in VISUAL [BALKIR 97]

§ Par « utilisation de requête mémorisée :
L'utilisateur interroge la base de données par l'intermédiaire de requête mémorisées dans une librairie.



Overlapping of icons in IconicBrowser

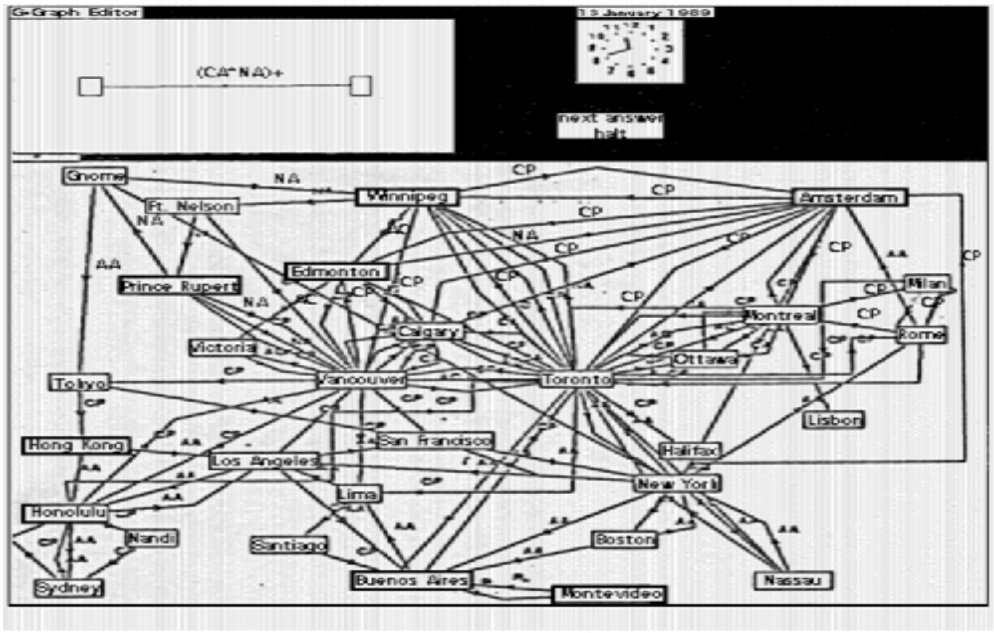
- **Par « Proposition » :**

Plusieurs représentations :

§ « Par proposition d'exemples » :
Ici, c'est plus une alternative à la formulation de requête traditionnelle (utilisateur propose la requête ensuite le système

retourne la réponse), l'utilisateur propose un exemple de résultats ensuite le système, par généralisation, identifie l'objectif pour répondre. La plus connue des applications (citée dans [CATARCI 95]°) est *QBE (Query By Example*).

§ « Par proposition d'un modèle » :
L'utilisateur utilise un modèle pour interroger le système. Ce dernier retourne toutes les solutions.



Pattern matching in G+

3.2.2 Critère 2 : Optimisation de la requête finale

L'optimisation utilise les mêmes moyens de représentation et de formulation de la requête. Son seul objectif est de permettre à l'utilisateur de pouvoirs optimiser sa requête.

Les différentes représentations des résultats sont par :

- Formulaire
- Diagramme
- Icône
- Hybride : Formulaire et Diagramme.

3.3 Classification des utilisateurs par rapport aux objectifs

Dans chacun des objectifs de la section précédente, l'utilisateur a une place importante. Celle-ci est identifiée par son but principal qui est :

Extraire l'information du système d'information, avec des requêtes performantes.

Nous pouvons dire que ces objectifs doivent satisfaire le but principal de l'utilisateur par rapport à son profil et à l'usage de l'information qu'il compte en faire.

Dans [CATARCI 95], l'analyse des profils utilisateurs a permis d'extraire quatre critères importants :

- La fréquence d'utilisation du VQS,
- La variance des requêtes,
- La complexité des requêtes,
- La connaissance de la sémantique des schémas de base de données.

Par rapports à ces critères, [CATARCI 95] donne les représentations de VQS appropriées :

		Icônes	Diagrammes	Formulaires
Fréquence d'utilisation	Fréquent		X	X
	Occasionnel	X		
Variance des requêtes	Répétitive		X	X
	Temps en temps	X		
Complexité des requêtes	Sophistiquées		X	
	Primaires	X		X
Connaissance des Schémas de BD	Familière		X	X
	Non-familière	X		

Tableau 1. Représentations de VQS appropriées

4 .Solution Idéale

[CATARCI 95] *L'utilisabilité est l'élément essentiel de la qualité de l'interaction entre l'utilisateur et le système en général.*

Ce chapitre présente la solution idéale du langage visuel d'interrogation de base de données pour notre problème. Cette solution n'a pas pour but dans un premier temps d'être réalisée mais de découvrir ce qui répondrait le mieux aux besoins de l'utilisateur sans soucier de la faisabilité technique. Pour cela nous allons étudier les caractéristiques d'un utilisateur du VQL pour en déduire la meilleure représentation, étudiée dans le chapitre précédent, qui lui correspond le mieux. Suite à cela nous présenterons notre solution en donnant une description de chacun des points de notre langage.

4.1 Caractéristiques de l'utilisateur :

Les interviews de l'analyse des besoins nous apportent les éléments nécessaires pour commencer à définir l'analyse de base du langage.

Nous savons qu'il existe un type d'utilisateur, concepteur de la requête, que nous définirons plus loin dans le document. Son travail consiste à analyser, optimiser, maintenir le process de l'aciérie et à en anticiper les problèmes.

Le concepteur est une personne qui est spécialiste dans son domaine. Son objectif principal est de récupérer le maximum de données dans le but de pouvoir réaliser des analyses, des statistiques ou des rapports. Ses connaissances dans son domaine d'expertise sont dues soit à son diplôme universitaire (ingénieur civil ou industriel), soit à son ancienneté dans la société et au fait d'avoir vu évoluer son « outil » en participant aux différentes rénovations. Ces deux méthodes d'apprentissage, de formation apportent vis à vis du VQL, l'avantage (constaté lors des interviews) que les concepteurs ont une connaissance de base en BD. Le fait qu'ils utilisent déjà l'outil informatique permet d'affirmer qu'ils ont une bonne dextérité avec la souris. Ils ont aussi une habitude des menus, des barres de tâches, des barres d'outils, et ils n'ont pas peur d'un cliquer-glisser.

L'avantage, pour le concepteur d'utiliser le VQL, est qu'il pourra être guidé dans sa réalisation de requêtes. VQL devra respecter l'objectif principal de l'utilisateur qui est de réaliser une requête pour lui ou ses subalternes dans un langage compris d'une base de données.

4.2 Choix de la représentation :

Le choix de la représentation devra permettre une utilisation au moins hebdomadaire, voir dans certains cas, quotidienne lors, par exemple, de recherche de solutions pour des problèmes bloquants la production continue. C'est-à-dire que le choix de la représentation devra aussi tenir compte de l'état psychologique de l'utilisateur (stress, pression…) Donc l'utilisation de l'outil d'interrogation n'est pas permanente, nous pouvons que la fréquence d'utilisation est moyenne.

Une bonne clarté dans le schéma et une représentation intuitive des requêtes permettront à l'utilisateur de réaliser la plus proche de ses idées, quelle que soit la situation de stress.

A l'heure actuelle, la complexité des requêtes est sophistiquée car ils utilisent les jointures et l'imbrication de requête SQL[21]. Ceci est dû au fait que leurs connaissances proviennent d'autoformation avec l'assistant disponible sur les produits Microsoft Access ou Microsoft Query. De ce fait, leurs connaissances en base de données sont limitées mais présentes.

[21] L'imbrication de requêtes est toutefois rare. Elle est utilisée par très peu de concepteurs.

La variance est moyenne puisque les concepteurs de requêtes utiliseront le plus possible celles existantes comme à l'heure actuelle et ne les changeront qu'en cas de demandes provenant de la Direction ou alors dans le cas d'optimisation du process.

Tableau des représentations les mieux appropriées pour le *concepteur* :

	Icônes	Diagrammes	Formulaires
Fréquence d'utilisation		X	
Variance des requêtes		X	
Complexité des requêtes		X	
Connaissance des Schémas de BD			X

Tableau 3.1. Représentations de VQS appropriées pour le Concepteur

Les choix des représentations ont été sur les critères définis au chapitre 2.
Les diagrammes se présentent comme étant les plus adéquats.

4.3 Spécificités de la représentation :

Pour représenter la réalité du centre d'intérêt de l'utilisateur, la présentation *Browsing Intensional* est très intéressante, car ne connaissant pas le schéma de la base de données (et ne le voulant pas), le *concepteur* pourra spécifier les concepts du process (par exemple : Coulées, Brames, Matières…) Le langage[22] l'aidera dans le choix des relations possibles avec d'autres éléments du domaine. Ceci présentera certains avantages pour un utilisateur novice qui pourra rechercher certaines informations de son domaine d'expertise et interroger la base de données comme il le souhaite avec les données et contraintes qu'il souhaite.

Pour formuler la requête, la *composition de concepts sur base de sous-requêtes*[23] sera la plus facile à manipuler par l'utilisateur. La taille de la base de données ne peut permettre les autres méthodes de formulation sans augmenter l'effort cognitif. Pour améliorer le déplacement dans la base de données, nous décidons d'adopter que se sera lors de la pose des concepts sur l'interface graphique que l'utilisateur pourra choisir les relations possibles avec d'autres concepts. Après avoir sélectionner un concept, une liste de relations s'affiche. L'utilisateur aura le choix des relations possibles avec d'autres. Par exemple, la sélection du concept « coulé » (posé au préalable sur l'interface graphique) permettra de visualiser une liste des relations possibles telles que : « appartient Plan de Coulées », « utilise Consigne », « demande Poches », … Toutes ces relations n'appartiennent qu'au concept coulée. Le choix d'une relation dans la liste permet de lier le concept à sa relation. La sélection de cette relation permet d'afficher à son tour la liste des concepts liés à celle-ci. C'est-à-dire dans l'exemple précédent, si on sélectionne la relation[24] « utilise consigne », on verra la liste des concepts liés avec celle-ci : « Ferraille », « Non Ferreux », « Ferreux ».

Cette séquence de sélections aura permis d'arriver par exemple à la représentation « Coulées utilisent les consignes Non Ferreux », déduite de la sélection lors de la dernière étape du concept « Non Ferreux » et de la lecture des éléments

[22] Cfr. Paragraphe 3.5
[23] Correspond dans le chapitre «Etat de l'art » par : « Formulation par sous-requête – composition de concept »
[24] sélectionnée au préalable dans la liste des relations possibles avec le concept « coulée »

« coulée », « Utilise consigne » et « Non Ferreux ». Cette représentation décrit un sous-ensemble de la base de données « Archivage » de CARSID.

La requête est réalisée au fur et à mesure des sélections dans les listes, donc la fin de celle-ci se fera à l'arrêt des sélections. Quant la requête est terminée, elle est appelée « requête finale ». En rapport avec le chapitre « état de l'art », celle-ci n'aura pas besoin d'être optimiser. Le VQL choisi dans ce chapitre aidera au fur et à mesure le concepteur dans la formulation de sa requête. Aucune représentation d'erreur ne sera nécessaire puisque le système fournit les relations à choisir par exemple Donc aucune erreur n'est possible.

4.4 Contrainte sur le langage

Le système de persistance cible est comme indiqué au chapitre 1, une base de données appelée « ARCHIVAGE ». Ce qui veut dire que la cible est une et une seule base de données.

4.5 Le Langage visuel

Le langage visuel servira lors de la création de la requête. Il est un moyen pour l'utilisateur pour exprimer son interrogation. Le résultat sera un affichage des données demandées dans un tableau. Le résultat contiendra toutes les valeurs demandées dans la requête.

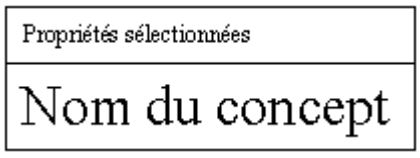
Il se compose de quatre représentations visuelles (graphiques) :

- Le domaine de requêtes formulées par l'utilisateur qui est constitué d'un ou plusieurs concepts mis en relation et qui appartiennent au domaine d'expertise de l'utilisateur.
- La relation qui lie des concepts
- L'opération qui est un traitement sur un ou plusieurs concepts
- La requête qui est la représentation d'un sous-ensemble de la base de données.

Il est basé sur le ***Browsing Intensional*** et la ***formulation par sous-requêtes***. C'est-à-dire que l'utilisateur compose sa requête avec les concepts de son monde d'expertise. Les relations possibles entre ses différents concepts lui seront affichées (imposés). Il pourra ajouter des opérations sur les concepts.

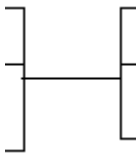
4.5.1 Le concept

Le concept de données se représente comme suit à l'utilisateur :



Le concept de données représente un ou plusieurs éléments du domaine d'expertise de l'utilisateur. Le concept est le lien entre le domaine de l'utilisateur et la base de données. C'est-à-dire que l'élément « Température capteur 200 » est tout à fait compris de l'utilisateur, mais dans la base de données il est représenté par CPT_TEMP_200. Dans ce cas simple, on peut dire que le concept est vu comme une traduction de la base de données. Par contre, un concept peut être en réalité un sous-ensemble du domaine. C'est-à-dire que le sous-ensemble « Prix moyen de toutes les pièces du parc AM02 de l'année 2002 du site Marcinelle » est aussi compris de l'utilisateur, mais ne correspond pas du tout à un seul champ de la base de la données. Ceci correspondra plus à une configuration préalable du système, dans le seul but de facilité le travail de l'utilisateur.

Le concept de données est un élément que l'utilisateur choisit sur l'interface graphique. Il n'a pas la possibilité d'en créer. Quand le concept est posé dans la zone de la requête de l'interface graphique, cela veut dire que l'utilisateur l'ajoute à sa requête. Si d'autres concepts ont déjà été sélectionnés, l'utilisateur devra le relier à ceux-ci. Pour cela une liste de relations ou d'opérations est mise à la disposition. Elle est spécifique au concept. Le lien avec une opération ou une relation sera marquée par un trait allant du concept à l'opération ou relation. Autre alternative via la liste des relations, accessible en sélectionnant le concept. La configuration de la sélection permet d'aboutir à la pose de la relation sur l'interface graphique et à la création du lien entre le concept et la relation.



Exemple de lien

Formalisme :

<concept> Nom
[<affichable/>]
[<dominant/>]
[<corrélation externe/>]
[<expressions/>]
</concept>

où

- Nom : désigne le nom du concept. (propriété obligatoire)
- affichable : Si présent alors le concept est un élément du sous-ensemble que l'utilisateur veut afficher. (propriété facultative)

Formalisme de la propriété affichable :

<affichable> [Nouveau Nom] </ affichable>

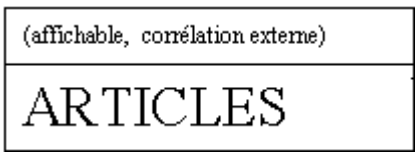
[Nouveau Nom] est facultatif, il désigne le nouveau nom du concept si l'utilisateur désire le changer.

- dominant : La dominance marquera les relations possibles avec les autres concepts ou requêtes. Si présent, alors le concept est dominant dans la requête. Pour comprendre la dominance, il faut savoir qu'une requête est un ensemble de concepts et de relations. Une relation relie uniquement deux concepts entre eux. Une requête peut être réutilisée par la suite dans une autre. Pour donner un sens à une requête, mais aussi pour résoudre le problème technique des relations, le langage demande qu'on choisisse les concepts dominants qui seront vus à l'extérieur dans le cas de requêtes imbriquées par exemple.(propriété facultative) (plus d'information sur 'dominant' dans le paragraphe 'Abstraction' du chapitre « Sémantique sur le langage »)

- corrélation externe : Si présent, le concept sera utilisable (accessible) en dehors de la requête. (propriété facultative) (plus d'information sur 'corrélation externe' dans le paragraphe 'Jointure' du chapitre « Sémantique sur le langage »)

- expressions : Elle contient l'expression arithmétique appliquée sur le concept. (propriété facultative) (cfr. ***Expressions arithmétiques sur les concepts***)

Exemple :

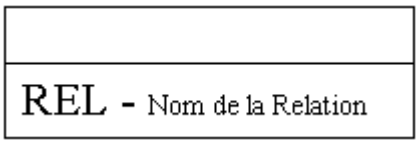


Formalisme de l'exemple :

<concept> ARTICLES <affichable/> <corrélation externe/> </concept>

4.5.2 La relation

La relation se représente à l'utilisateur comme suit :



Une relation est un lien conceptuel entre deux concepts. Elle est accessible via une liste globale ou via la liste spécifique des concepts.

Sa signification est symbolique au sein d'une requête, dans le sens où elle ne permet que de lier des concepts entre eux et qu'elle ne soit pas un élément du domaine proprement dit.

Son rôle est de permettre de voyager de concepts en concepts dans le domaine et d'accéder ainsi à une représentation des idées de l'utilisateur. Le nom des relations devra donc être explicite pour pouvoir deviner les concepts directement accessibles.

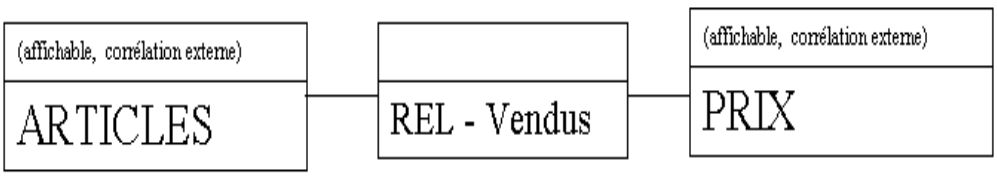
Formalisme :

<relation> Nom </relation>

où

- Nom : désigne le nom de la relation. (propriété obligatoire)

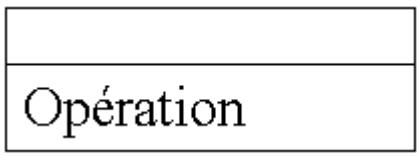
Exemple :



Formalisme de l'exemple :
<concept> ARTICLES <affichable/> <corrélation externe/> </concept>
<relation> Vendus </relation>
<concept> PRIX <affichable/> <corrélation externe/> </concept>

4.5.3 L'opération

La relation se représente à l'utilisateur comme suit :



Une opération représente un traitement quelconque sur un ou plusieurs concepts. L'opération ne pourra être réalisée qu'entre types compatibles, c'est-à-dire des entiers avec des entiers, des nombres décimaux avec des entiers ou des décimaux, etc.

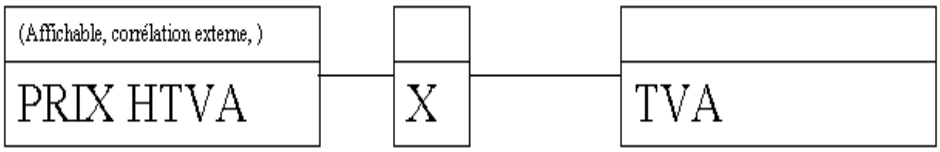
Formalisme :

<opération> Nom </opération>

où

- Nom : désigne le nom de l'opération. (propriété obligatoire)

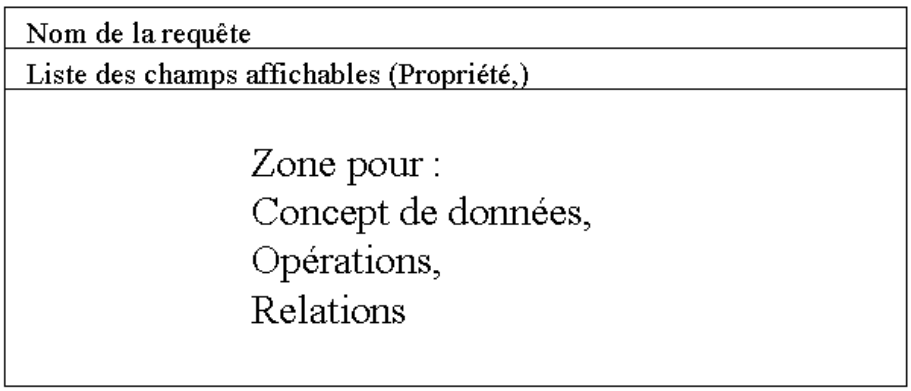
Exemple :



Formalisme de l'exemple :
<concept> PRIX HTVA <affichable/> <corrélation externe/> </concept>
< opération > X </ opération >
<concept> TVA </concept>

4.5.4 La requête

La requête se représente comme suit à l'utilisateur :



Une requête définit un sous-ensemble du domaine de l'utilisateur. Pour décrire ce domaine, elle a besoin des concepts de données, de pouvoir effectuer des opérations sur ces mêmes concepts et de les relier par des relations.

Deux concepts non reliés par une relation ou une opération ne peuvent exister dans une même requête. Ceci est dû au fait qu'une requête définit un sous-ensemble du domaine et non plusieurs sous-ensembles.

Une requête aboutira à une interrogation sur la base de données. De ce fait, elle englobe les concepts à afficher, les critères d'interrogation (filtres, contraintes, …), les relations entre les concepts. Une requête peut-être utilisée par une autre en tant que concept. Dans ce cas, il faudra définir le ou les concepts « dominant ». La dominance marquera les relations possibles avec les autres concepts ou requêtes.

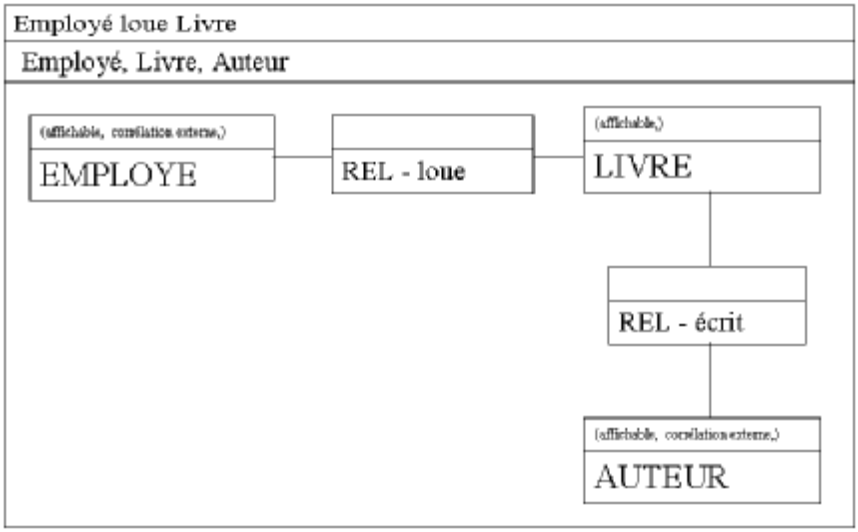
Formalisme :

<requête> Nom [<non redondant/>] <concept/> </ requête >

où

- Non redondant : Lors de l'affichage du résultat, les valeurs associées aux concepts « affichable » et apparaissant plusieurs fois ne seront reprise qu'une et seule fois.

Exemple :



Formalisme de l'exemple :
<requête> Employé loue livre
<concept> EMPLOYE <affichable/> <corrélation externe/> </concept>
< relation > loue </ relation >
<concept> LIVRE </concept>
< relation > écrit </ relation >
<concept> AUTEUR </concept>
</ requête>

4.5.5 Sémantique du langage

Dans les pages qui suivent nous parcourons les différents points de la sémantique du langage. Nous montrerons comment les différents aspects du langage permettent de décrire des sous-ensembles d'un domaine. Tout ceci dans le but de permettre à l'utilisateur de convertir sa pensée en requête vers une base de données.

Le formalisme complet du langage est expliqué ci-dessous sous forme de grammaire :

Remarques :

- *Les propriétés facultatives sont indiquées entre crochets ([]).*
- *Les propriétés comprises entre []* précisent qu'on peut les répéter de 0 à plusieurs fois.*

<requete/> :

<requete> Nom [<non redondant>] <concept/> </requete>
| <requete> Nom [<non redondant>] <requetes/>[<opération/><requete/>]</requete>*
| <requete> Nom [<non redondant>] <requetes/>[<join/><requete/>]</requete>*

<concept/> :

<entité concept/>
*| <entité concept/> [<relation/> <entité concept/>]**
*| <entité concept/> [<opération/> <entité concept/>]**
*| <entité concept/> [<opération/> <Numérique/>]**

<entité concept/> :

<entité concept> Nom
[<affichable/>]
[<corrélation externe/>]

[<expression/>]
[<dominant/>]
[<minimum/>]
[<maximum/>]
[<moyenne/>]
[<compter/>]
[<like/>]
</entité concept>

<join/> :

<concept externe/> <opération/> <concept externe/>

<concept externe/> :

<concept externe> NOM REQUETE . NOM CONCEPT </concept externe>

<opération/> :

<opération> Nom </opération>

<relation/> :

<relation> Nom </relation>

<Numérique/> :

<numérique> Valeur numérique </ numérique >

<expression/> :

<expression> expression arithmétique </ expression >

<like/> :

<like> syntaxe standard SQL du like </ like >

<dominant/> :

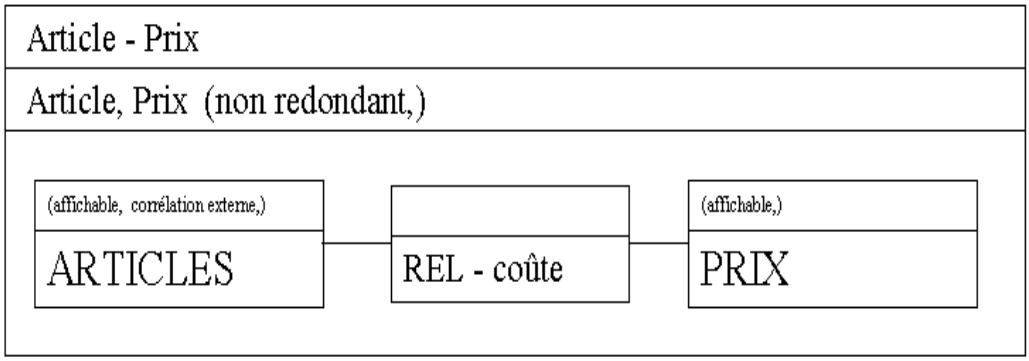
<dominant > nom du concept dominant </ dominant >

- **Projection**

La projection permet de n'afficher que certains concepts d'un domaine. Si on ne veut pas afficher les doublons alors il faut ajouter la propriété « non redondant ». La technique est simple et utilise les méthodes vues ci-dessus.

Une projection se compose au minimum d'un concept de données et peut regrouper des concepts reliés entre eux par des relations.

Exemple :



Cette requête se traduit en :

Afficher la liste des articles avec leurs prix, sans doublon sur les paires Articles et Prix.

Formalisme de l'exemple :

<requete>Article-Prix
<concept>ARTICLES<affichable/><corrélation externe/></concept>
<relation>coûte</relation>
<concept>PRIX<affichable/></concept>
</requete>

Equivalent en SQL :

```
SELECT DISTINCT ARTICLES, PRIX
FROM OFFRE
```

- **Jointure**

Une jointure est une opération permettant de combiner des informations de plusieurs requêtes. Pour cela les concepts permettant la jointure doivent avoir la propriété « corrélation externe ». Dès ce moment, ils sont accessibles à l'extérieur donc joignables par une opération

La jointure est créée en associant deux requêtes par une opération. Celle-ci a comme attaches deux concepts de chaque requête.

Formalisme :

<requête><requête/><join/><requête/></requête>

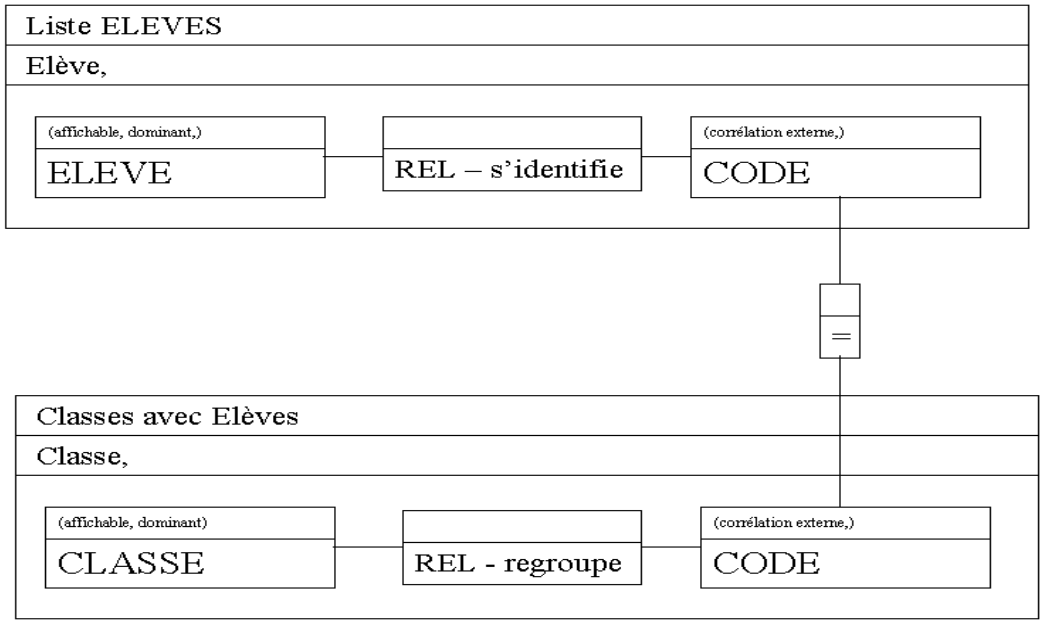
où

- <requête/> : représente une requête telle que vue précedement
- <join/> :représente le formalisme de la jointure. Celle-ci se formule comme suit :

<join><concept externe><opération/><concept externe></join>

Un concept externe est un concept possédant la propriété « corrélation externe » dans la requête précédant et suivant le <join/>

Exemple :



Cette requête se traduit en :

Afficher la liste des élèves associés à leurs classes.

Formalisme de l'exemple :

```
<requete >_
<requete>Liste élèves
<concept>ELEVE <affichable/><dominant/>
</concept>
        <relation>s'identifie</relation>
<concept>CODE <corrélation externe/>
</concept>
</requete>
<join>  <concept externe> Liste élèves.CODE</concept externe>
        <operation>=</opération>
        <concept externe> Classes avec élèves.CODE<concept externe>
</join>
<requete>Classes avec élèves
<concept>CLASSE<affichable/><dominant/>
</concept>
        <relation>regroupe</relation>
<concept>CODE <corrélation externe/>
</concept>
</requete>
</requete>
```

Equivalent en SQL :

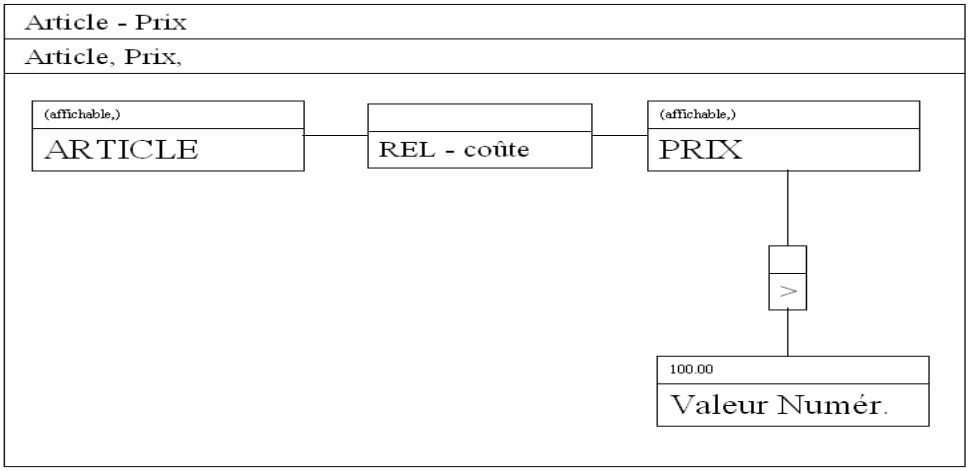
```
SELECT NOM_ELEVE, INTITULE_CLASSE
FROM ELEVE, CLASSE
WHERE ELEVE.CODE = CLASSE.CODE
```

- **Sélection**

La sélection permet de définir un sous-ensemble du domaine en lui appliquant des contraintes.

La sélection (associée à la jointure) est la technique de base la plus courante employée par les utilisateurs. Elle correspond à la méthode la plus simple pour créer des requêtes. C'est-à-dire qu'à partir d'une projection, pour définir une sélection, il suffit d'ajouter une opération. Ceci dans le but de réduire le domaine du résultat. Cette contrainte sera toujours liée au départ à un concept.

Exemple :



Cette requête se traduit en :

Afficher la liste des articles qui coûtent plus de 100 €.

Formalisme de l'exemple :

<requete>Article-Prix
<concept>ARTICLE <affichable/></concept>
<relation>coûte</relation>
<concept>PRIX <affichable/></concept>
<opération> > </opération>
<Numérique>100.00</Numérique>
</requete>

Equivalent en SQL :

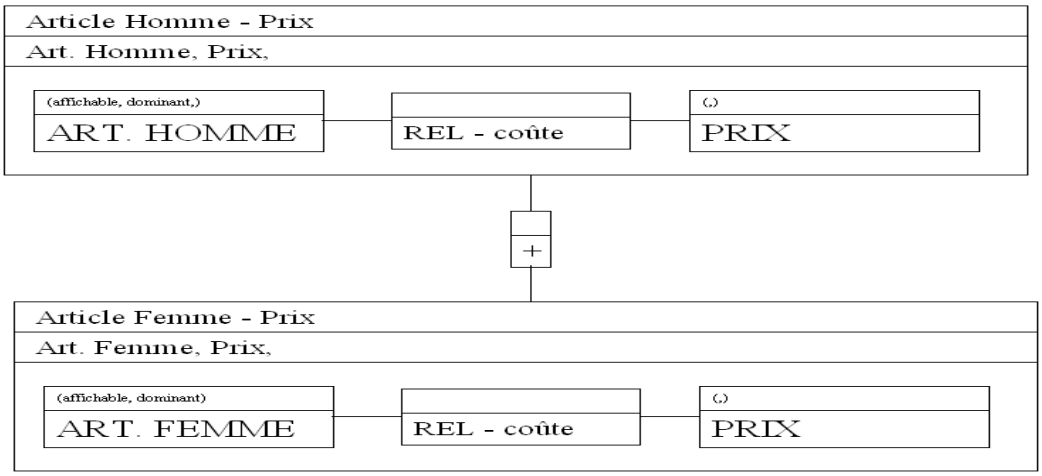
```
SELECT ARTICLE, PRIX
FROM OFFRE,
WHERE PRIX >100.00
```

- **Union**

L'union de deux requêtes est un sous-ensemble contenant chaque élément de la première requête *et* chaque élément de la seconde. Les éléments communs aux deux requêtes ne sont conservés qu'en un seul exemplaire, c'est-à-dire que l'opération d'union élimine les doublons. Les concepts que l'on fait correspondre dans les deux requêtes n'ont pas besoin de porter les mêmes noms.

La technique se présente par une opération '+' qui joint les 2 requêtes.

Exemple :



Cette requête se traduit en :

Afficher la liste de tous les articles hommes et femmes.

Formalisme de l'exemple :

```
<requete>_
<requete>Art. Homme - Prix
<concept>ART. HOMME<affichable/><dominant/></concept>
        <relation>coûte</relation>
<concept>PRIX</concept>
</requete>
<operation>+</opération>
<requete> Art. Femme - Prix
<concept>ART. FEMME<affichable/><dominant/></concept>
        <relation>coûte</relation>
<concept>PRIX</concept>
</requete>
</requete>
```

Equivalent en SQL :

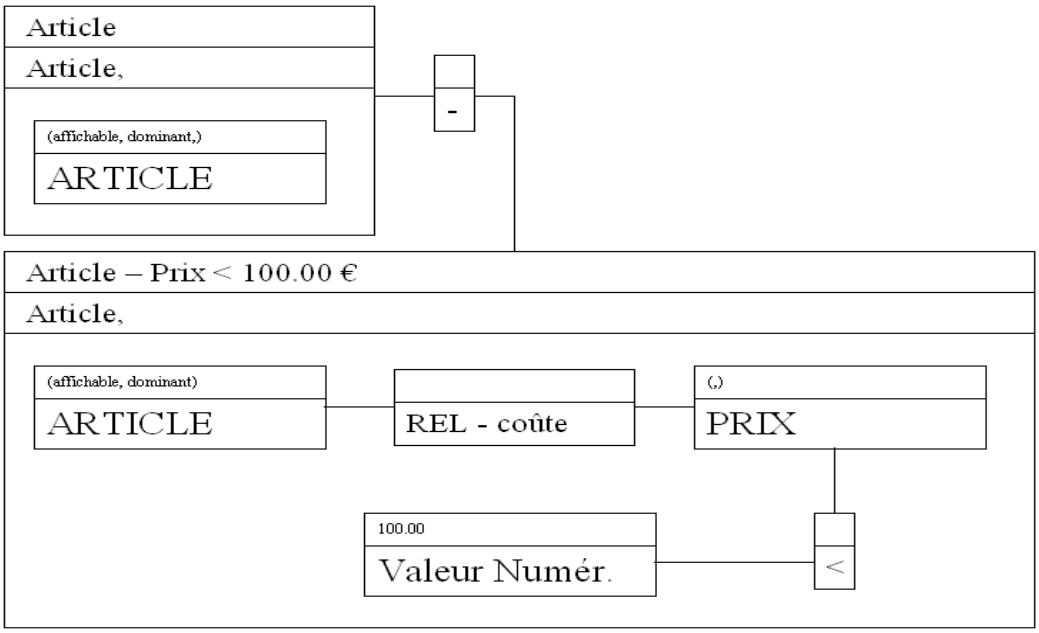
```
SELECT HOMME.ARTICLE
FROM HOMME
UNION
SELECT FEMME.ARTICLE
FROM FEMME
```

- **Différence d'ensembles**

La différence de deux requêtes est un sous-ensemble contenant les éléments de la première requête qu'on ne retrouve pas dans la seconde. Les conditions sont les mêmes que pour l'union.

La technique se présente en associant les 2 requêtes par une opération de soustraction (signe '-').

Exemple :



Cette requête se traduit en :

Afficher la liste de tous les articles – ceux qui coûtent moins de 100.00€.

Formalisme de l'exemple :

```
<requete>_
<requete>Article
<concept>ARTICLE<affichable/><dominant/></concept>
</requete>
<operation>-</opération>
<requete> Article – Prix < 100.00€
<concept>ARTICLE<affichable/><dominant/></concept>
        <relation>coûte</relation>
<concept>PRIX</concept>
        <opération><</opération>
<numérique>100.00</numérique>
</requete>
</requete>
```

Equivalent en SQL :

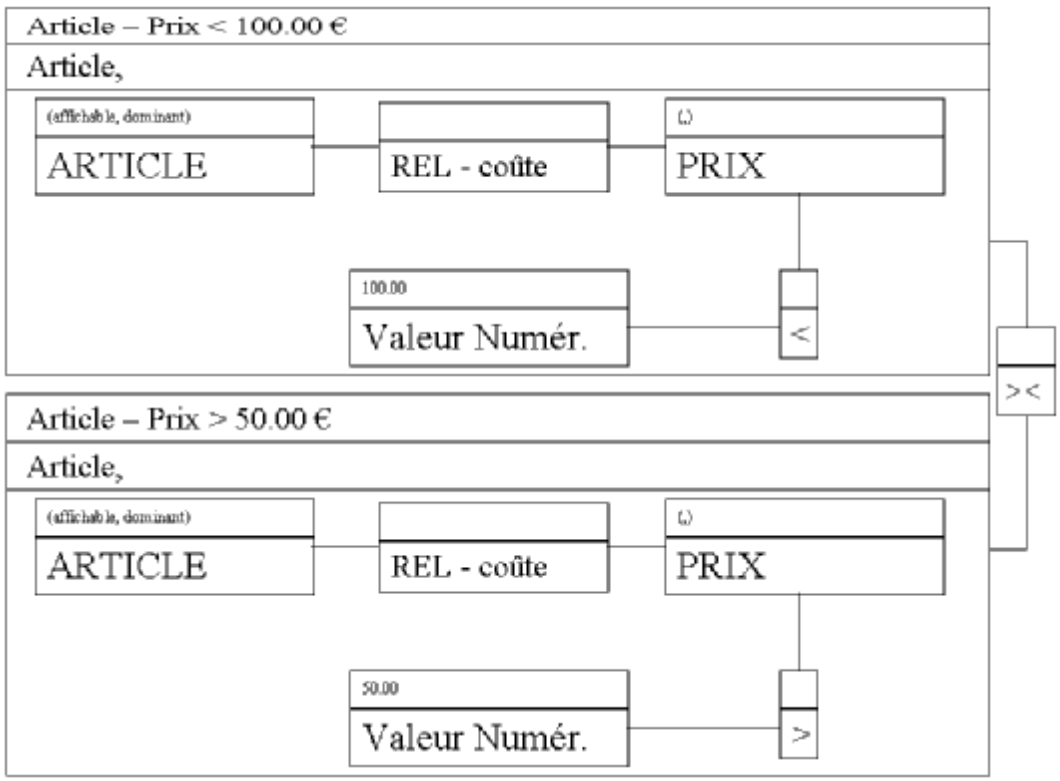
```
SELECT ARTICLE
FROM OFFRE
EXCEPT
SELECT ARTICLE
FROM OFFRE
WHERE PRIX < 100.00
```

- **Intersection**

L'intersection de deux requêtes est un sous-ensemble contenant seulement les éléments communs aux deux requêtes. Les conditions sont les mêmes que pour l'union.

La technique se présente en ajoutant l'opération '><' et en reliant aux 2 requêtes.

Exemple :



Cette requête se traduit en :

Afficher la liste de tous les articles compris entre 50.00 et 100.00€.

Formalisme de l'exemple :

<requete>_
<requete> Article – Prix < 100.00€
<concept>ARTICLE<affichable/><dominant/></concept>
<relation>coûte</relation>
<concept>PRIX</concept>
<opération><</opération>
<numérique>100.00</numérique>
</requete>
<operation> >< </opération>
<requete> Article – Prix > 50.00€
<concept>ARTICLE<affichable/><dominant/></concept>
<relation>coûte</relation>
<concept>PRIX</concept>
<opération> > </opération>
<numérique>50.00</numérique>
</requete>

</requete>

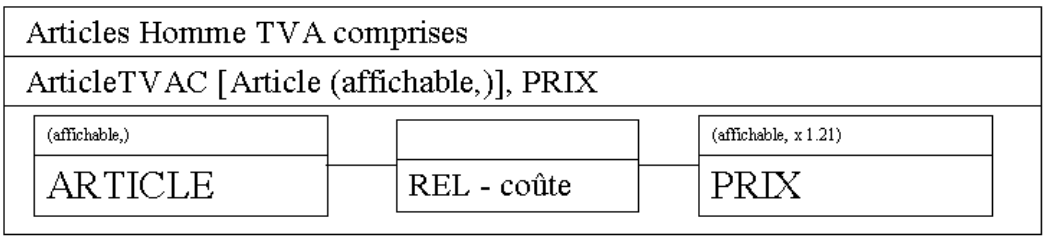
Equivalent en SQL :

```
SELECT ARTICLE
FROM OFFRE
WHERE PRIX <100.00
INTERSECT
SELECT ARTICLE
FROM OFFRE
WHERE PRIX > 50.00
```

- **Expressions arithmétiques sur les concepts**

Permet des calculs sur les concepts tels que addition, multiplication, … Nous avons la possibilité de renommer le concept à afficher en donnant le nouveau nom et en l'associant au concept dans la requête qu'on précise entre crochets (avec toutes ses propriétés).

Exemple :



Cette requête se traduit en :

Afficher les articles homme TVA comprise.

Formalisme de l'exemple :
<requete>Article Homme TVA comprises
<concept>ARTICLE <affichable/></concept>
<relation>coûte</relation>
<concept>PRIX <affichable/><expressions> x 1.21 </expressions>
</concept>
</requete>

Equivalent en SQL :

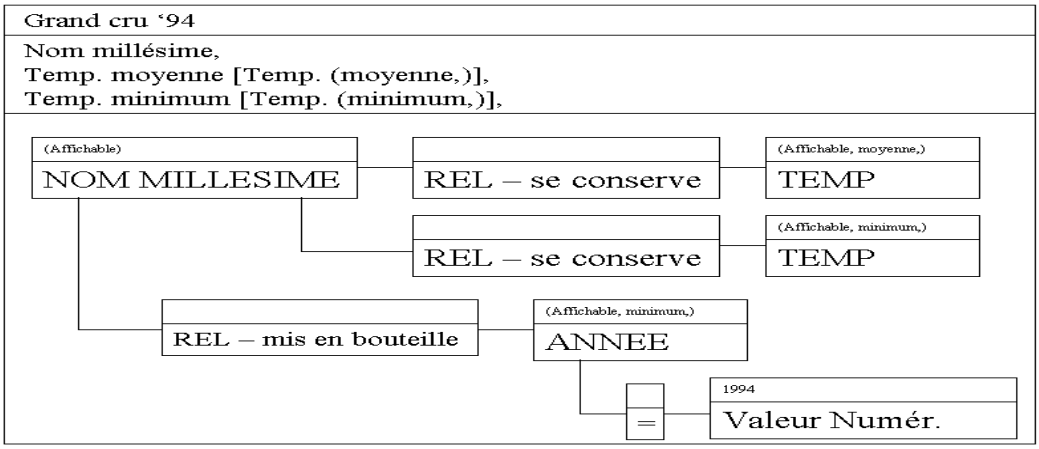
```
SELECT PRIX*1.21
FROM HOMME
```

- **Agrégats**

Derrière cette appellation se cachent des fonctions permettant de faire du dénombrement, déterminer le minimum, maximum, … Ces opérateurs sont une extension très significative de l'algèbre relationnelle. Pour les utiliser, il suffit de les insérer dans les propriétés du concept concerné.

Opérateurs possibles :Compter, Moyenne, Maximum, Minimum.

Exemple :



Cette requête se traduit en :

Calculer le degré moyen et le degré minimum pour tous les crus de '94

Formalisme de l'exemple :

```
<requete>Grand cru '94
<concept>NOM MILLESIME <affichable/></concept>
        <relation>se conserve</relation>
<concept>TEMP <affichable/><moyenne/></concept>
<concept>NOM MILLESIME <affichable/></concept>
        <relation>se conserve</relation>
<concept>TEMP <affichable/><minimum/></concept>
<concept>NOM MILLESIME <affichable/></concept>
        <relation>mis en bouteille</relation>
<concept>ANNEE <affichable/><minimum/></concept>
        <opération>=</opération>
<numérique>1994</numérique>
</requete>
```

Equivalent en SQL :

```
SELECT CRU, AVG(DEGRE), MIN(DEGRE)
FROM VINS
WHERE MILLESIME = 1994
GROUP BY CRU
```

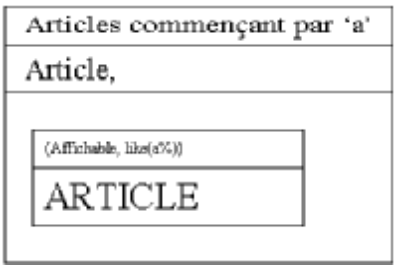
- **Filtre sur une chaîne de caractères**

Permet de filtrer des chaînes de caractères en utilisant comme syntaxe la règle standard de SQL appliquée pour « *like* », c'est-à-dire :

% : tient lieu de 0 ou plus d'un caractère arbitraire.

_ :est utilisé pour un caractère manquant.

Exemple :



Cette requête se traduit en :

Afficher les articles commençant par 'a'.

Formalisme de l'exemple :

<requete>Article commençant par 'a'
<concept>ARTICLE <affichable/><like>a%</like></concept>
</requete>

Equivalent en SQL :

```
SELECT ARTICLE
FROM HOMME
WHERE ARTICLE LIKE 'a%'
```

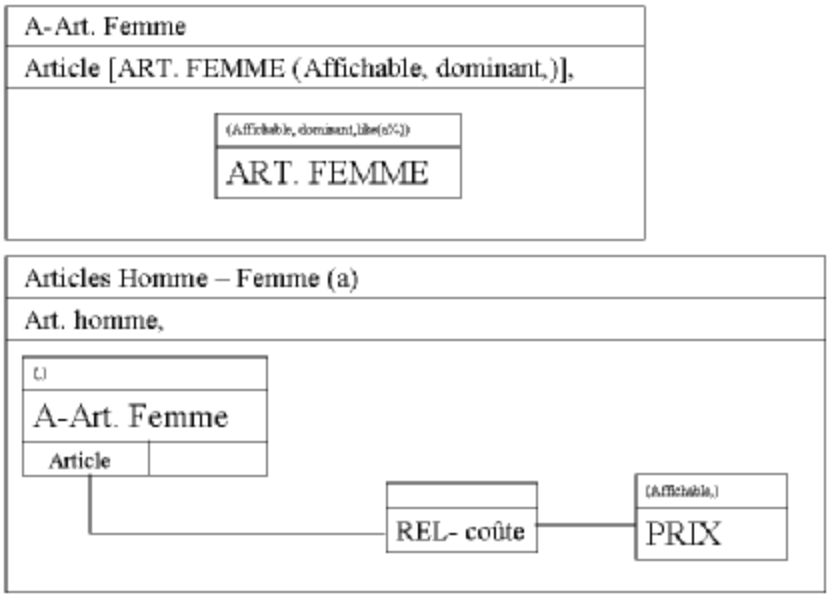
- **Abstraction**

L'abstraction permet d'inclure une requête dans une autre avec les mêmes propriétés qu'un concept. Ceci permet de donner plus de clarté au schéma et d'améliorer la lecture de la requête. La requête minimisée est une requête qui possède des relations avec d'autres concepts. Celles-ci ont pour origine le ou les concepts qui dominent dans le sens apporté à la requête.

La technique à employer est de définir une requête tel que vue précédemment. Cette requête devra posséder des liens avec d'autres concepts. Pour les connaître, il suffit de prendre les concepts qui apportent du sens à la requête (généralement ceux qu'on décide d'afficher) et on leur attribue la propriété 'dominant'. A partir de ce moment, les concepts apporteront leurs relations à la requête, mais en plus ils seront vu à l'extérieur de celle-ci. C'est-à-dire dès qu'on posera le symbole de la requête minimisée dans la requête utilisatrice, on voit qu'en plus du nom, on a aussi les concepts définis précédemment comme dominant. Ceci permet de les sélectionner et d'avoir accès

à la liste de leurs relations. Ensuite les manipulations sont connues, si non se référer aux paragraphes précédents.

Exemple :



Cette requête se traduit en :

Afficher les articles femmes commençant par 'a' et leurs prix.

Formalisme de l'exemple :

```
<requete>A-Art. Femme
        <concept>ART. FEMME
        <affichable>Article</affichable>
        <like>a%</like>
        <dominant/>
        </concept>
</requete>
<requete> Articles Homme – Femme(a)
<requete> A-Art. Femme <dominant>Article</dominant></requete>
<relation> coûte </relation>
<concept>PRIX<affichable/></concept>
</requete>
```

Equivalent en SQL :

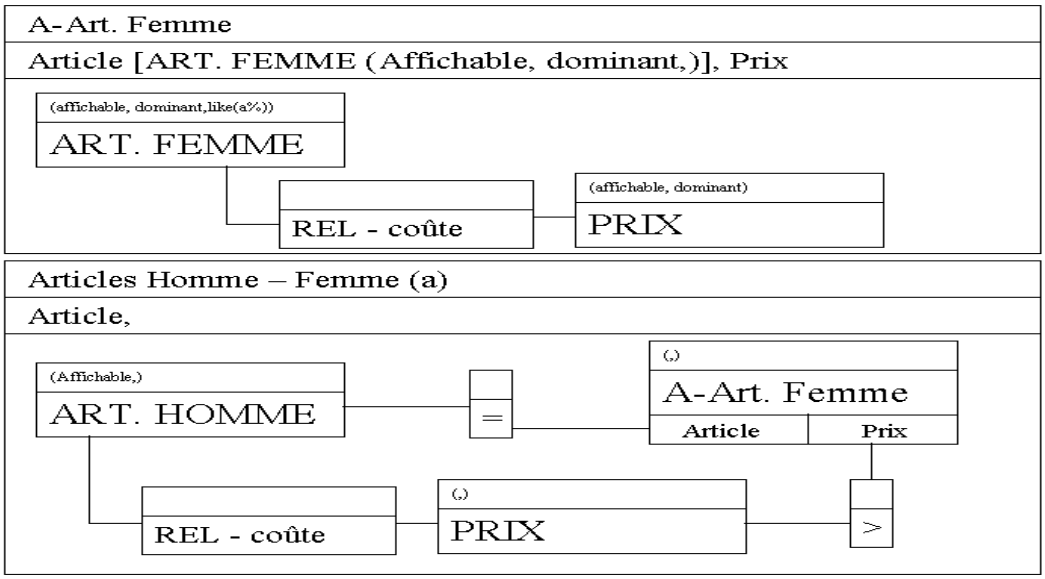
```
SELECT H.ARTICLE
FROM HOMME AS H,
(SELECT F.ARTICLE FROM F WHERE F.ARTICLE LIKE 'A%') as FA
WHERE H.ARTICLE = FA.ARTICLE
```

- **Requêtes imbriquées**

Il est possible d'utiliser le résultat d'une requête, dite imbriquée, comme condition de recherche d'une autre requête.

Celles-ci utilisent la notion d'abstraction ci-dessus, mais font apparaître la notion des concepts dominants à l'extérieur des requêtes pour qu'ils soient utilisés par des opérations.

Exemple :



Cette requête se traduit en :

Afficher les articles homme commun à ceux des femmes commençant par 'a', et ayant un prix supérieur à ceux des articles femmes.

Formalisme de l'exemple :

```
<requete>A-Art. Femme
        <concept>ART. FEMME
                <affichable>Article</affichable>
                <like>a%</like>
                <dominant/>
        </concept>
                <relation>coûte</relation>
        <concept>PRIX
                <affichable/>
                <dominant/>
        </concept>
</requete>
<requete> Articles Homme – Femme(a)
<concept>Art. Homme<affichable/></concept>
        <opération> = </opération>
<requete> A-Art. Femme <dominant>Article</dominant></requete>
<concept>Art. Homme<affichable/></concept>
        <relation> coûte</relation>
<concept> PRIX </concept>
        <opération> > </opération>
<requete> A-Art. Femme <dominant>Prix</dominant></requete>
```

</requete>

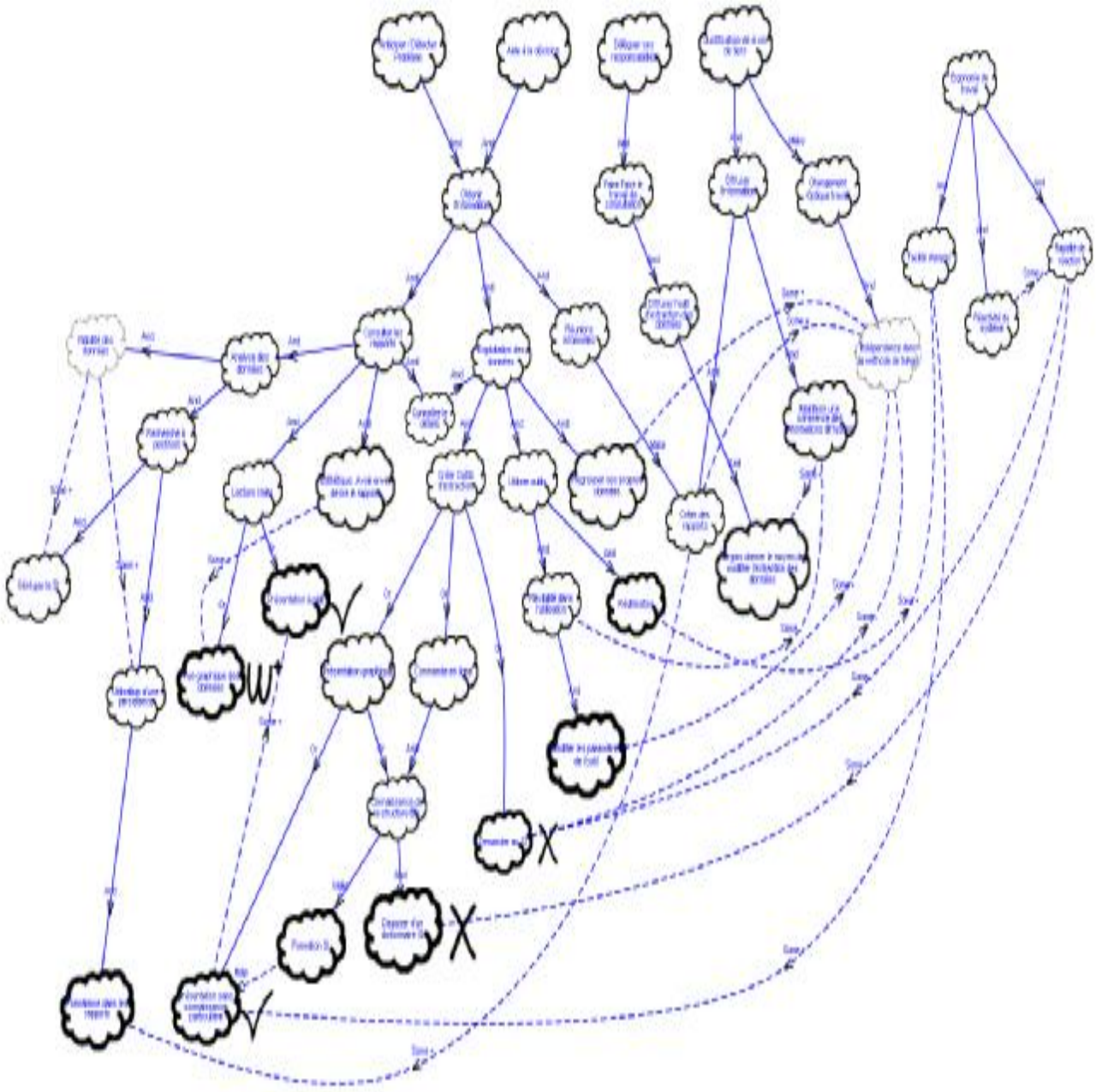
Equivalent en SQL :

```
SELECT H.ARTICLE
FROM HOMME AS H,
(SELECT F.ARTICLE, F.PRIX FROM F WHERE F.ARTICLE LIKE
'A%') as FA
WHERE H.ARTICLE = FA.ARTICLE
AND H.PRIX > FA.PRIX
```

5 .Solution réduite

5.1 Analyse des besoins

5.1.1 Arbre des buts



5.1.2 Objectifs fonctionnels

Les objectifs fonctionnels auxquels doivent répondre le système à réaliser sont :

- Formalisme dans les rapports
- Outils d'extraction :
  - Le résultat de la requête doit être sous forme « plate »
  - La création de la requête ne doit pas nécessiter de connaissance particulière sur la structure de la base de données
- Modification des paramètres de l'extraction en mode utilisateur
- Réutilisabilité des requêtes
- Regroupement des données

5.1.3 Objectifs non fonctionnels

Les objectifs non fonctionnels auxquels doit répondre le système à réaliser sont :

- Facilité d'emploi
- Formations données par le SI pour que l'utilisateur apprenne les rudiments de l'outil.
- Réactivité du système : il doit pouvoir fournir une manipulation fluide et non lente.
- Envoi des rapports géré par le SI.

5.1.4 Profils utilisateurs

Pour ce paragraphe, nous avons repris dans la partie « users » du document « User Check-list »[25] les différents points permettant de connaître le profil d'un acteur du système.

Deux types d'acteurs sont identifiés dans l'analyse des besoins. Nous appellerons le premier comme étant un « Concepteur » et le deuxième comme étant un « Utilisateur »

- **Concepteur**

  - **Attributs Physiques**
    - Pas d'âge, ni de sexe spécifique.
  - **Attributs Mentaux**
    - Pas de handicap mental fort permis.
  - **Qualifications et connaissances**
    - Doit avoir une connaissance parfaite de son domaine. Le concepteur est soit un ingénieur responsable d'un secteur de l'aciérie ou alors un contremaître en chef. Il doit savoir utiliser un ordinateur, manipuler une souris et connaître la notion du « cliquer-glisser » sur Windows.
    - Doit connaître le français.
  - **Caractéristiques du travail**

[25] produit par Monique Noirhomme et Anne deBaenst

- **Fonction :** Créer des requêtes sur la base de données de l'aciérie et les diffuser.
- **Fréquence :** Utilisation courante de l'outil.
- **Flexibilité :** 5jours / semaine et pendant les heures de bureau et.

q Utilisateurs

- **Attributs Physiques**
  - Pas d'âge, ni de sexe spécifique.
- **Attributs Mentaux**
  - Pas de handicap mental fort permis.
- **Qualifications et connaissances**
  - Aucune qualification générale n'est requise. Il doit savoir utiliser un ordinateur, manipuler une souris et connaître la notion du « copier-coller » sur Windows.
  - Doit connaître le français.
- **Caractéristiques du travail**
  - **Fonction :** Exécuter les requêtes fournies par le *Concepteur* et exporter le résultat vers une autre application.
  - **Fréquence :** Utiliser de manière répétitive l'outil.
  - **Flexibilité :** 5jours / semaine et pendant les heures de bureau.

5.1.5 Conclusion

Objectif de l'utilisateur :

Interroger une base de données sans connaître la manière dont celle-ci est structurée.

Objectif Technique :

Réaliser un outil pour interroger une base de données via un langage visuel.

Objectif du langage visuel :

Interroger une base de données via les concepts du domaine.

- **Spécificité de l'outil :**

- Présentation du résultat à plat.
- Interface intuitive (graphique) et sans connaissance particulière.
- Réutilisabilité de l'interrogation.
- Possibilité de regroupement des données.
- Avantager la distribution de l'interrogation, sans perte de cohérence (mode utilisateur/ concepteur)

Cet outil devra être :

- **Facile d'emploi**
- **Réactif**

- **Autre spécificité :**

- Le système de persistance sera géré par le SI.
- Le SI formera les utilisateurs.

5.2 Architecture Conceptuelle

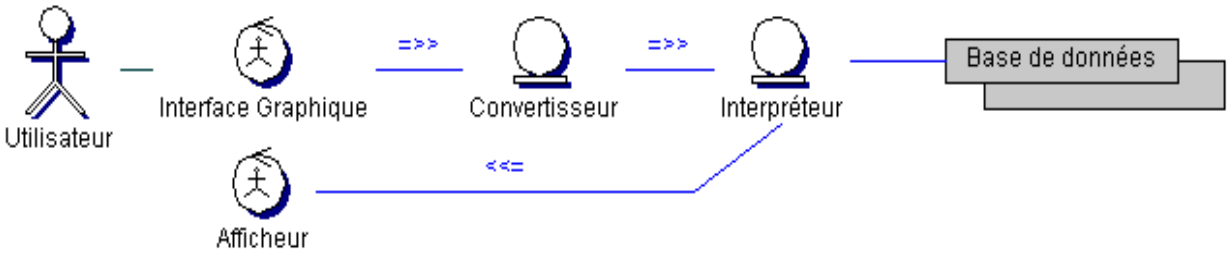
Objectif technique :

L'analyse des besoins nous apporte comme contrainte la réalisation d'une interface graphique qui permettra de réaliser des requêtes pour interroger une base de données.

Choix de l'architecture :

Type de cohérence : sur base d'un diagramme de flux.

Cette architecture est basée sur un ensemble de composants indépendants. Chaque intervenant ne connaît pas les autres en aval/amont d'où un meilleur couplage. Le travail réalisé ne dépend pas de l'ordre dans lequel les composants sont associés.



Composant Interface graphique (GUI) :

Il reçoit les questions de l'utilisateur et les convertit en une structure de données manipulable par un ordinateur.

Composant Convertisseur :

Il convertit la structure de données représentant les questions de l'utilisateur en un langage de Base de données.

Composant Interpréteur :

Il exécute la requête de l'utilisateur sur la Base de données. La réponse de la base de données est retournée dans un afficheur.

Composant Afficheur :

C'est un simple tableau affichant un ensemble de données. Celui-ci sera le résultat de la requête.

L'interpréteur et l'afficheur peuvent être un seul outil tel que Excel, Access ou MS Query. Ces outils sont fortement répandus dans toutes les sociétés.

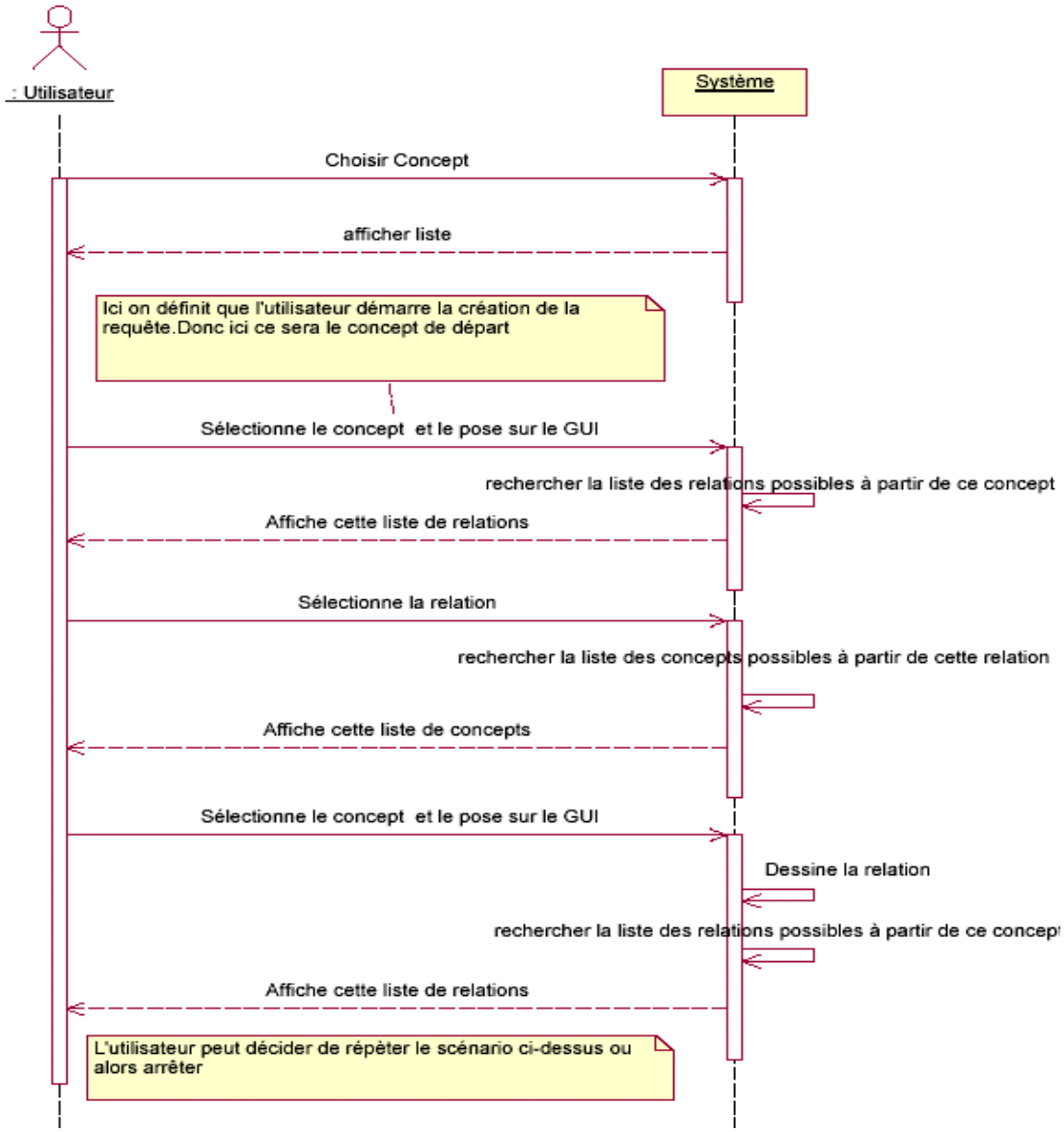
Le convertisseur devra interpréter la structure des données (arbre syntaxique par exemple) correspondant au schéma réalisé par l'utilisateur sur le GUI. La technique employée pour générer un code (langage) sur base d'une structure de données a été vue au cours de Monsieur Schobbens et lors du TP compilateur. Cette technique est maîtrisée (cfr livre Dragon) et ne posera pas de problème. Cette partie n'est pas le scope du mémoire)

L'interface graphique (GUI) restera ma principale recherche dans la suite de ce chapitre.

5.3 Architecture logique abstraite de l'interface graphique

L'objectif principal sera de s'approcher le plus possible de la solution idéale. Le point de vue conceptuel est facile à voir mais essayons d'approcher le point de vue technique de cette solution et réfléchissons pour voir comment en aborder la réalisation.

La solution idéale a comme scénario d'utilisation celui décrit ci-dessous :



On s'aperçoit dans ce scénario de la présence d'une dynamique de proposition de liste soit de concepts, soit de relations. On sait aussi que l'utilisateur devra choisir un concept de départ pour sa requête.

La solution idéale définit des propriétés sur les concepts qui seront importantes lors de la construction du SQL. Certaines propriétés sont cependant aussi importantes pour l'ergonomie de la requête : par exemple le filtre sur un concept, les agrégats… L'élément « requête » de la solution idéale possède lui-aussi des propriétés, telles que son nom, la liste des concepts affichés dans le résultat, la liste des concepts utilisables dans les autres requêtes…

Après ce bref aperçu, quatre éléments se présentent comme étant les éléments principaux de l'interface graphique : la requête, le concept, la relation et les propriétés. Chacun de ces éléments devra être défini dans l'architecture et devra y trouver sa place.

La relation et le concept seront des éléments graphiques que l'utilisateur devra manipuler. Par contre la requête et les propriétés seront des éléments que l'utilisateur devra définir. Ceux-ci ont deux sens différents afin de définir qui est responsable de quoi.

Nous définissons donc un repository[26] qui apportera à l'utilisateur l'information sur les relations et les concepts. L'utilisateur par ailleurs apportera l'information sur la requête et les propriétés. Ces deux règles permettent de séparer les responsabilités de chacun, c'est-à-dire :

- Le repository sera responsable de l'affichage des relations et des concepts.
- L'utilisateur sera responsable des définitions de la requête et des propriétés liées aux concepts.
- L'utilisateur sera responsable du nom de la requête et des propriétés liées à celle-ci.

Ci-dessous le schéma de collaboration entre le Repository et l'utilisateur à travers les concepts, les relations, les propriétés et la requête :

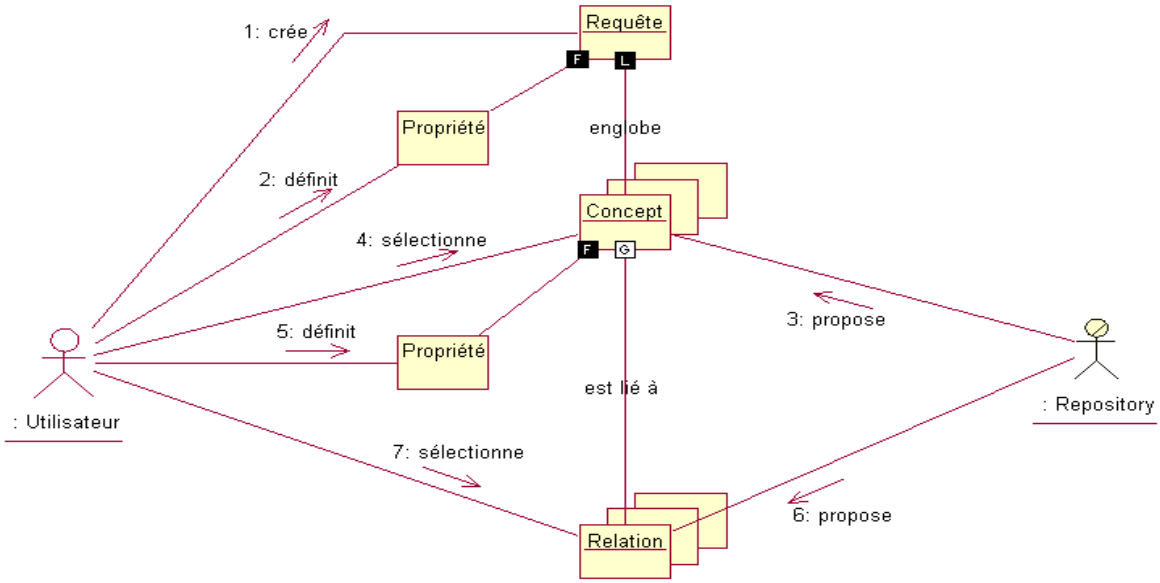
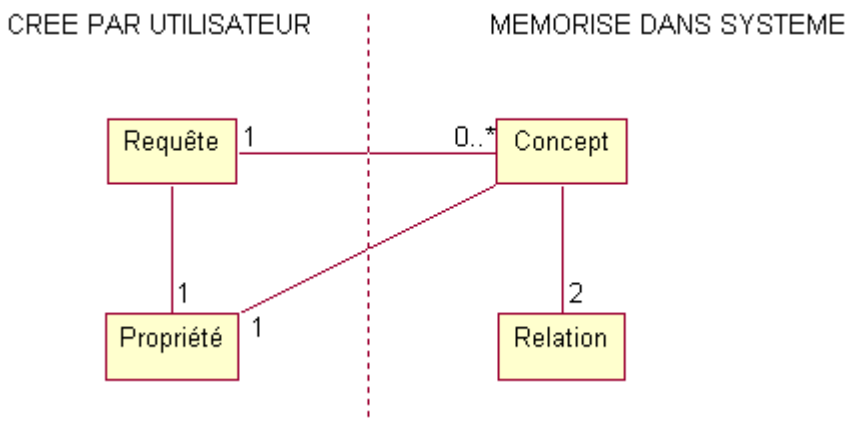
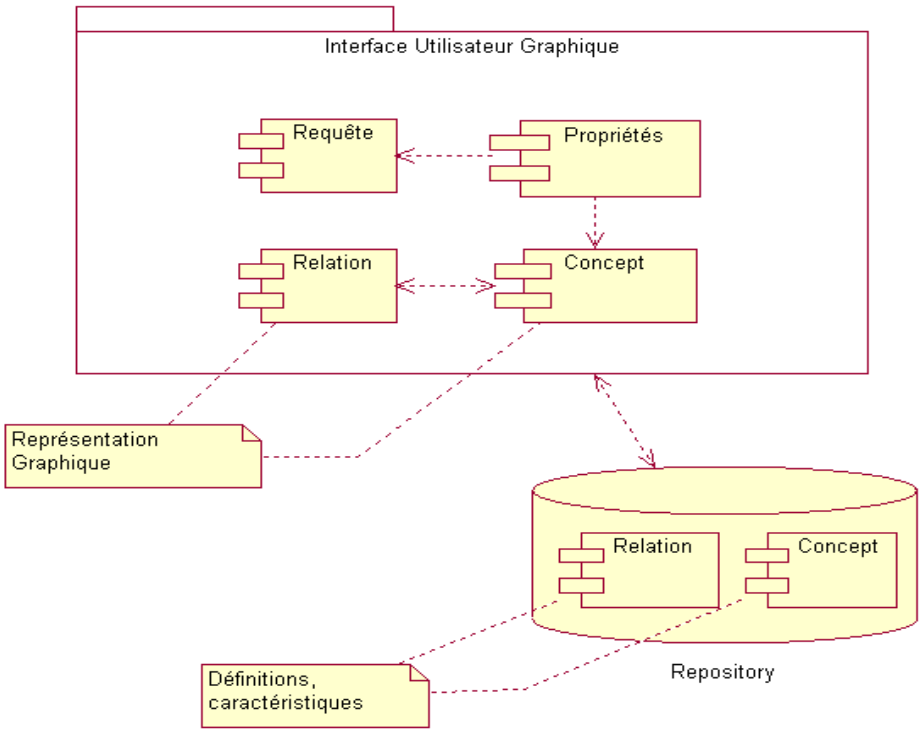


Diagramme contextuel déduit du diagramme de collaboration :



[26] Définition : référentiel, système de persistance.

Architecture logique abstraite



Le système GUI du VQS peut se présenter ainsi : un Interface Utilisateur Graphique et un Repository. L'interface permettra à l'utilisateur de manipuler les composants Requête, Propriétés, Relation et Concept. Le repository apportera les spécificités aux composants Relation et Concept.

Définitions des composants :

- Relation : Composant graphique. Sa signification pour l'utilisateur sera de relier deux concepts. D'un point de vue sémantique le fait que l'utilisateur pourra ou non mettre cette relation entre deux concepts sera défini dans le repository.
- Concept : Composant graphique. Il représentera pour l'utilisateur un élément atomique ou non du domaine d'expertise. Les concepts seront contenus dans le repository.
- Requête : Composant graphique. L'utilisateur utilisera ce composant contenant les concepts et les relations de la requête.
- Propriétés : Composant non graphique lié au composant requête et au composant concept. Il apparaîtra à l'utilisateur au moment où celui-ci voudra ajouter des caractéristiques spécifiques à chacun des composants posés sur l'interface.

5.4 DIAGEN

L'interface graphique à réaliser doit communiquer avec l'utilisateur pour lui permettre de réaliser sa requête, puis ensuite la convertir en une structure cohérente et manipulable par un système informatique. Ce type d'interface existe sous l'appellation de générateur de diagramme. Celui qui sera utilisé est connu sous le nom de DIAGEN[27].

Le texte qui suit est traduit des documents *The Tree Tutorial* et *Overview* de DIAGEN écrit par l'université de Erlangen.

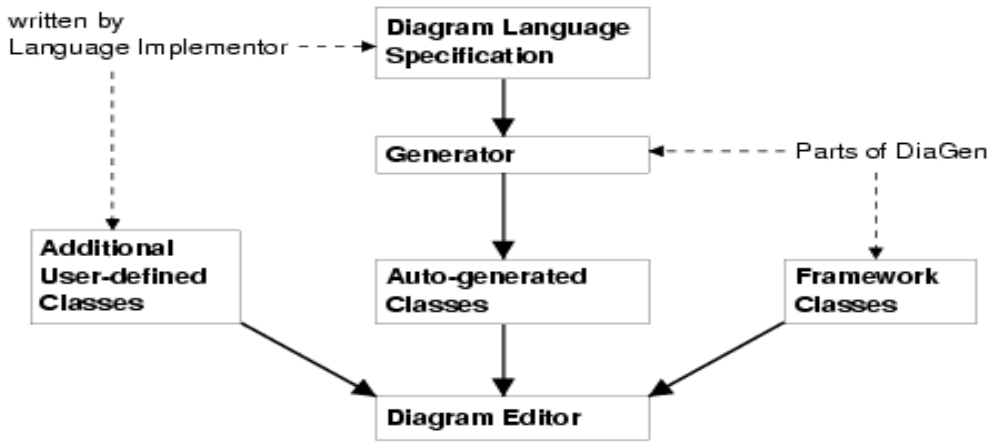
5.4.1 Présentation

Diagen est un système pour faciliter le développement de puissant éditeur de diagramme. Il se compose de 2 parties :

- Un framework de classes java qui donne accès à des fonctionnalités génériques pour manipuler et analyser des diagrammes.
- Un générateur de programmes pour produire du code Java pour la plupart des fonctionnalités qu'on retrouve dans la sémantique des diagrammes.

5.4.2 Caractéristiques principales du système

Pour définir un langage spécifique de diagramme et pour créer un générateur, le concepteur du langage doit donner des spécifications formelles du langage du diagramme[28]. Ces spécifications sont traitées par le générateur, qui produit des classes Java qui étendent les classes de base du framework. Elles spécifient aussi la structure syntaxique du langage visuel. Pour que l'éditeur soit complet, le concepteur du langage doit ajouter du code Java qui relie l'affichage graphique et la dynamique des diagrammes.



*Le système **DIAGEN***

L'ensemble des caractéristiques principales suivantes distingue *Diagen* des autres systèmes d'édition et d'analyse de diagrammes existants sur le marché :

- L'éditeur de *DiaGen* inclut un module d'analyse pour identifier la structure et l'exactitude syntaxique des diagrammes en cours de

[27] DIAGEN signifie ***Dia**gram editor **gen**erator :*
http://www2-data.informatik.unibw-muenchen.de/DiaGen/
[28] Appelé dans la suite du document : langage visuel

manipulation. L'analyse structurale est basée sur les transformations et les grammaires d'hypergraphe, qui fournissent un modèle syntaxique flexible et autorise une analyse efficace. DiaGen a été conçu pour être insensible à l'analyse et la manipulation des diagrammes qui sont seulement partiellement corrects.

- *DiaGen* emploie les résultats de l'analyse structurale pour une syntaxe de haut niveau et une facile interaction avec le *layout*. Le mécanisme du *layout* est basé sur des contraintes géométriques flexibles et compte sur un moteur de résolution externe.

- *DiaGen* combine une édition à main levée avec un programme de dessin qui permet les principales modifications structurales d'un diagramme. Le concepteur du langage peut fournir facilement des opérations qui seront prises en compte par l'éditeur lors de la création du diagramme. Ces opérations sont des options puissantes qui sont généralement là pour soutenir des tâches d'édition fréquentes, mais elles ne doivent pas perturber tous les besoins d'édition qui pourraient être prévues.

- *DiaGen* est entièrement écrit en Java et est basé sur Java 2 SDK. Il est donc réalisé sur une plateforme indépendante du système d'exploitation et peut profiter pleinement de tous les dispositifs des nouvelles API[29] graphiques de Java2D : Par exemple, DiaGen soutient sans restriction le zoom, et le rendu de la qualité est ajusté automatiquement pendant les interactions de l'utilisateur.

[29] API signifie Application Program Interface
Définition : An application program interface (API - and sometimes spelled *application programming interface*) is the specific method prescribed by a computer operating system or by an application program by which a programmer writing an application program can make requests of the operating system or another application.

5.4.3 Aperçu de l'architecture

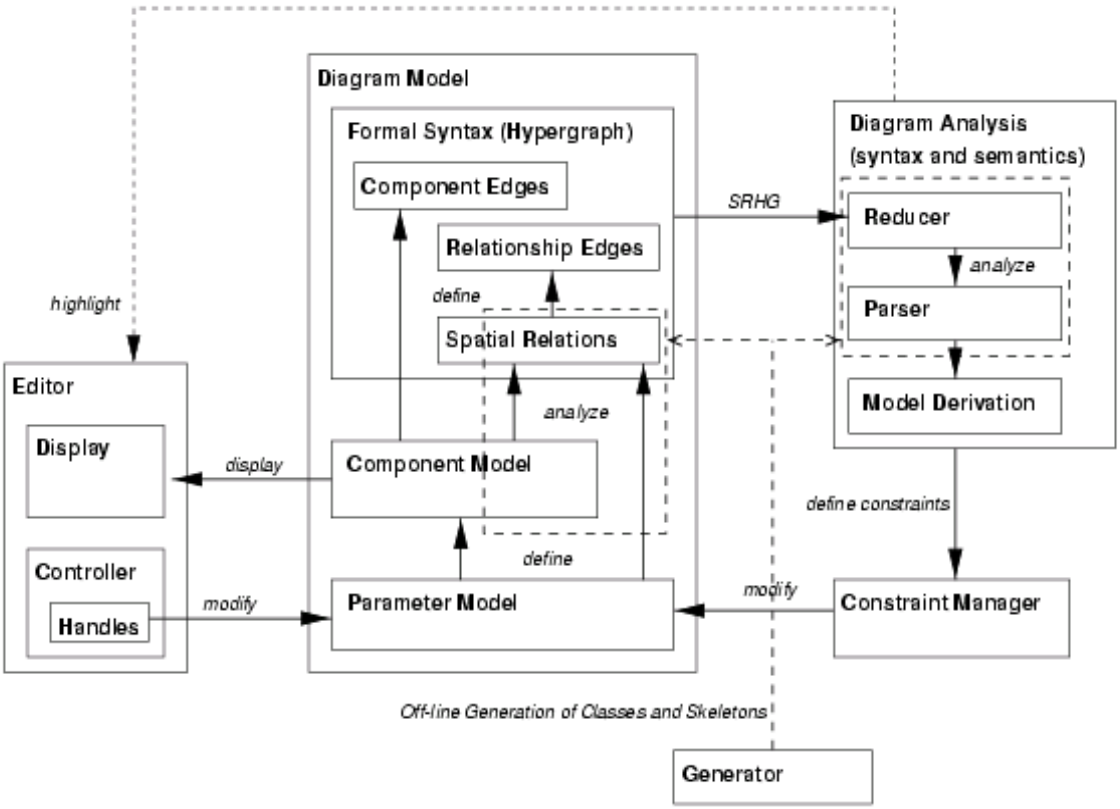


Figure 2: Architecture of an editor generated by and based on *DiaGen*

La figure ci-dessus esquisse l'architecture d'un éditeur qui a été produit par le générateur de *DiaGen* et qui est basé sur le framework de *DiaGen.* Le diagramme montre les différents modules qui composent l'éditeur et les flux de données qui les lient entre eux. Les sections suivantes présenteront la structure de cette figure et sa terminologie spécifique. Cette présentation examinera d'abord la structure imposée par le framework de *DiaGen* qui alimente les majeures parties de l'éditeur. Ensuite, les spécialisations qui sont nécessaires pour créer un éditeur à partir du framework seront décrites brièvement.

La figure indique les composants du framework à adapter aux besoins du cas. On les situe grâce au générateur de code source de DiaGen qui effectue les majeures parties du procédé de personnalisation.

5.4.4 Le modèle de diagramme

Le composant principal de l'architecture du framework est le modèle de diagramme. Au cours de l'exécution du programme, la représentation du diagramme est sous forme d'objets situés en mémoire. Le terme « Modèle » est employé ici dans le sens du paradigme de Modèle-Vue-Contrôleur[30] et doit être distingué des modèles abstraits qui peuvent être dérivés par l'analyse de plus haut niveau du diagramme. Le modèle de diagramme se compose :

[30] Traduit de Model – View – Controler. Ici *Diagen* est basé sur la version 1 de ce paradigme.

- D'un ensemble de composants qui représentent le diagramme (par exemple des cercles, des lignes ou des objets complexes comme les classes UML et leurs définitions)
- Et leur représentation dans la syntaxe formelle d'hypergraphe ;
- Il inclut également une description du type de diagrammes qui peuvent être manipulés en cours d'exécution.

Le modèle de diagramme est divisé en trois couches :

1) Au niveau le plus bas, le modèle est représenté comme une collection de `` paramètres``, de simples nombres réels qui déterminent les propriétés des composants de diagramme. Par exemple, à ce niveau la représentation d'un cercle pourrait être :

xcenter1 = 10, ycenter1 = 12, radius1 = 5

Une paire de paramètres forme souvent un point, par exemple.

center1 = (xcenter1, ycenter1)

Chaque paramètre appartient à exactement un composant ; les changements du modèle se produisent toujours au niveau des paramètres d'abord et sont ensuite propagés vers le haut.

2) Le modèle composant décrit comment la représentation graphique du diagramme est calculée à partir des paramètres. Il implémente aussi un flux d'évènements qui met à jour la représentation graphique quand les paramètres changent. La représentation graphique est alimentée par cette couche qui utilise les classes et les concepts du Java2D API.

3) La couche formelle de syntaxe est établie sur le modèle composant. DiaGen emploie des hypergraphes comme représentation de plus haut niveau des diagrammes : chaque composant a un ou plusieurs *secteurs* d'attachement, secteurs sensibles qui peuvent interagir avec d'autres composants. Pour une flèche, ces secteurs seraient habituellement ses points finaux, alors que pour un cercle se pourrait être son centre. Chaque secteur d'attachement forme un nœud dans le modèle d'hypergraphe ; un composant est représenté par un hyperedge qui relie tous les noeuds de ses secteurs d'attachement.

La manière dont les composants sont combinés pour former un diagramme représente les *relations spatiales*. Ce sont des relations basées sur les secteurs d'attachement qui sont définis par des attributs sur les paramètres des composants correspondants. Par exemple, deux cercles pourraient avoir la relation « toucher » si les paramètres de leurs composants satisfont l'équation :

distance(center1, center2) = radius1 + radius2

Actuellement, l'éditeur graphique soutient seulement les rapports binaires qui devraient en fait être suffisants pour tous les cas pratiques.

Chacun de ces rapports spatiaux trace une relation dans le modèle d'hypergraphe qui relie les noeuds correspondant aux secteurs d'attachement. La méthode employée par DiaGen appelle la représentation formelle résultante de la syntaxe du diagramme *spatial relationship hypergrah* (SRHG). Ce dernier se compose d'un ensemble de composants et de noeuds distincts attachés à eux, qui sont alors reliés par des relations.

Les couches de modèles forment une hiérarchie dans le sens que chaque couche peut seulement dépendre des couches inférieures ; par exemple, il serait possible de changer la représentation formelle de syntaxe sans affecter le modèle de composants et de paramètres.

5.4.5 Syntaxe et sémantique

Le SRHG fourni par le modèle de la syntaxe formelle est traité par le module d'analyse syntaxique et sémantique. L'architecture de DiaGen coupe l'analyse de diagramme en trois étapes :

1) Le « réducteur » simplifie le SRHG par le remplacement des sous-graphes, qui décomposent certains modèles, avec des structures plus simples. Les règles de réduction peuvent introduire des contextes négatifs ainsi nous pouvons ajouter des conditions additionnelles qui doivent être mémorisées avant qu'une étape de réduction soit exécutée. Dans le résultat simplifié, appelé le modèle d'hypergraphe (HGM[31]), tous les secteurs d'attachement qui sont reliés par des relations spatiales significatives devraient être transformés et toutes les relations qui n'ont pas de signification sémantique devraient être éliminées.

2) Le HGM peut alors être analysé par un *incremental hypergrah parser*[32]. L'utilisation des hypergraphes au lieu des graphiques standards permet habituellement d'exprimer la plupart des conditions structurales d'une langue typique de diagramme sous une forme hors contexte.

3) Les informations résultantes sur la structure du diagramme peuvent alors être employées pour établir un modèle sémantique à niveau élevé du contenu du diagramme. La forme concrète de cette représentation sémantique est fortement dépendante de l'application et peut donc ne pas être définie dans le framework. Une possibilité pour l'usage de cette information sémantique est de traduire le contenu du diagramme en un autre (par exemple en SQL).

5.4.6 L'édition dirigée par la syntaxe

Le fait que DiaGen emploie un modèle formel du diagramme en interne (le SRHG) permet d'aider l'opération d'édition à un niveau élevé qui opère comme à un niveau abstrait. C'est-à-dire que des modifications spécifiques correspondant à une interprétation complexe du diagramme peuvent être décrites comme des transformations d'hypergraphe sur le SRHG. L'utilisateur peut choisir une telle opération ainsi que les composants qui indiquent un contexte partiel d'application dans le SRHG ; le système peut alors automatiquement étendre le contexte comme indiqué pour la transformation et l'exécuter. Le SRHG modifié est alors analysé encore et, avec l'aide du mécanisme automatique basé sur les contraintes mises à disposition, il est transformé de nouveau dans une représentation visuelle.

De telles opérations à niveau élevé donnent à l'utilisateur des moyens d'exécuter facilement de plus grands changements structurels sur le diagramme. Par exemple, des sous-structures peuvent être débranchées, déplacées à un autre endroit et être rebranchées là presque automatiquement dans un pas à pas. Les équipements d'édition orientée syntaxe complètent, en effet, seulement l'édition à main levée. Les opérations d'édition n'ont pas besoin de permettre toutes les modifications nécessaires. Elles servent seulement de sténographies à quelques cas précis tandis que l'édition à main levée peut encore être employée pour réaliser les modifications de diagramme que le programmeur de l'éditeur de diagramme n'a pas prévues.

---

[31] *hypergraph model*

[32] *hypergrah parser* emploie les techniques semblables au CYK-algorithm standard pour des grammaires de chaîne de caractères.

5.5 Architecture logique concrète de l'interface graphique

Le repository

Elément de persistance contenant toutes les relations et tous les concepts du domaine d'expertise de l'utilisateur. C'est-à-dire qu'il contient au moins le domaine conceptuel ou le schéma conceptuel de la base de données à consulter.

Le repository sera une base de données relationnelle.

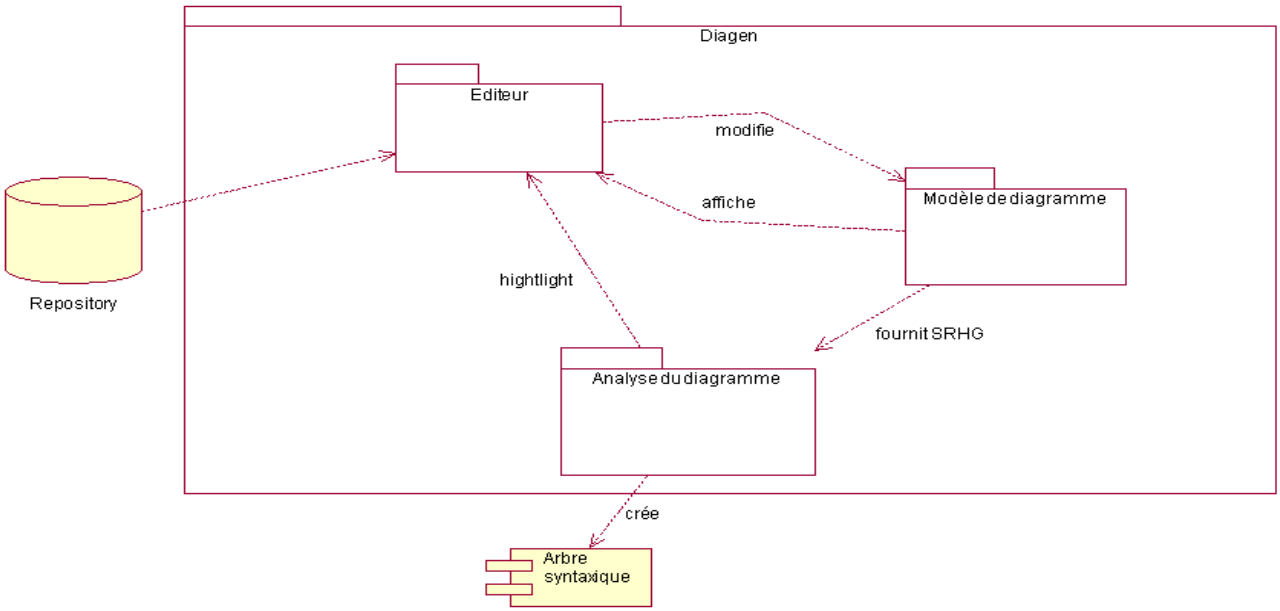
L'interface Utilisateur Graphique

L'utilisateur va consulter, sélectionner et poser sur l'interface les composants graphiques qui représenteront sa requête. L'interface sera en relation avec le repository au moment où l'utilisateur devra sélectionner un Concept ou une Relation.

L'interface sera réalisée avec l'éditeur de DiaGen. La sémantique sera incorporée par le modèle de composant et donc définit par l'hypergraph (syntaxe formelle)

Sur base de DiaGen, l'interface construira un arbre syntaxique au fur et à mesure de la construction de la requête. Dès la fin de celle-ci, à partir de cet arbre, un compilateur pourra le transformer en SQL tel que vu au cours de *syntaxe et sémantique*[33].

Diagramme de composant de l'architecture logique concrète :



Dans ce diagramme nous avons volontairement supprimé le composant « Constraint manager[34] » de l'architecture de DiaGen. Celui-ci intervient dans le cas où l'utilisateur voudrait simplifier le diagramme construit. Ceci n'intervient pas dans le cadre de l'outil à réaliser.

Deux nouveaux composants apparaissent le Repository et l'Arbre syntaxique. Ce dernier est implémenté par défaut dans DiaGen mais pour comprendre où il se situe, nous l'avons extrait de l'architecture. Ce composant devra être accessible par le compilateur soit par l'intermédiaire d'un fichier, soit par un composant sérialisable. Pour comprendre le fonctionnement de l'analyse

[33] Cours dispensé aux FUNDP. Professeur : Monsieur Schobbens
[34] cfr. §4.3.6

du programme, le lecteur pourra se référer au § 4.3.5 : *Syntaxe et Sémantique*. Le repository est lié à l'éditeur pour que celui-ci puisse le consulter à la demande et surtout pour que la consultation ne soit pas dépendante du diagramme. Ceci permettra par exemple à l'utilisateur de pouvoir consulter les relations d'un composant juste en posant celui-ci sans se préoccuper de la syntaxe du diagramme.

5.6 Réalisation du repository

Le repository a pour objectif de fournir au GUI les concepts et les relations du domaine conceptuel ou du schéma conceptuel de la base de données cible.

C'est le service SI qui a en charge d'encoder les valeurs et de les maintenir dans le repository. Les informaticiens de ce service sont les seuls compétents pour lier le schéma conceptuel à la base de données. Car le lien entre le schéma conceptuel, c'est-à-dire les valeurs contenues dans le repository, et le schéma de la base de données est fait dans une autre base de données. Ce lien conceptuel-physique est utilisé au niveau de la conversion de la question de l'utilisateur en requête SQL. Cette conversion intervient au niveau du composant convertisseur représenté dans l'architecture conceptuelle. Ce point sera expliqué dans le paragraphe « Perspectives futures ».

Schéma conceptuel

Une relation est comprise entre deux concepts. Son sens va du concept d'origine au concept de fin, c'est-à-dire:

concept origine => relation => concept de fin.

Schéma :

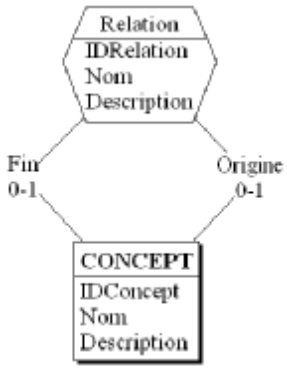


Schéma logique

Règle : Une relation n'est pas obligée d'être liée à un concept, mais si c'est le cas alors les deux doivent exister.

Schéma :

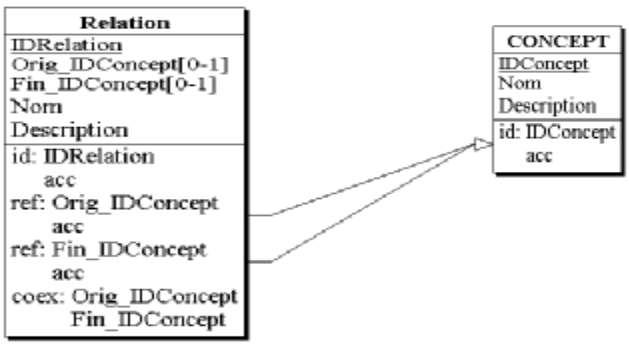
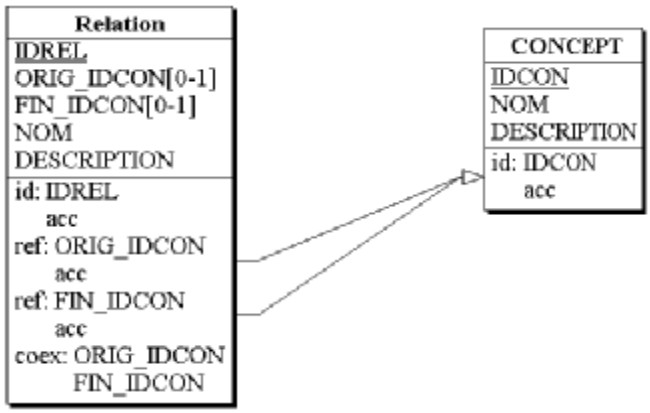


Schéma physique



SQL généré avec DB-MAIN

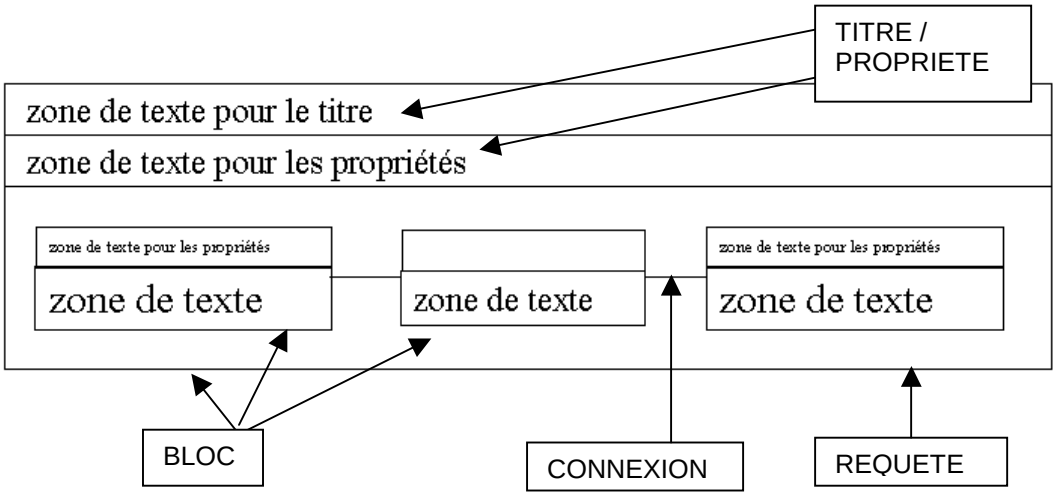
```
-- ********************************************
-- * Standard SQL generation                  *
-- *------------------------------------------*
-- * Generator date:  Nov  6 2002             *
-- * Generation date: Sat May 01 11:41:28 2004 *
-- ********************************************


-- Database Section
-- ________________

create database PHYSIQUE;


-- DBSpace Section
-- ________________


-- Table Section
-- _____________

create table CONCEPT (
     IDCON char(5) not null,
     NOM varchar(1) not null,
     DESCRIPTION varchar(1) not null,
     primary key (IDCON));

create table Relation (
     IDREL char(5) not null,
     ORIG_IDCON char(5),
     FIN_IDCON char(5),
     NOM varchar(1) not null,
     DESCRIPTION varchar(1) not null,
     primary key (IDREL));


-- Constraints Section
-- ___________________

alter table Relation add constraint FKOrigine
     foreign key (ORIG_IDCON)
     references CONCEPT;

alter table Relation add constraint FKFin
     foreign key (FIN_IDCON)
     references CONCEPT;

--alter table Relation add constraint GRRelation
--     check((ORIG_IDCON is not null and FIN_IDCON is not null)
--           or (ORIG_IDCON is null and FIN_IDCON is null));


-- Index Section
-- _____________
```

```
create unique index IDCONCEPT
     on CONCEPT (IDCON);

create unique index IDRelation
     on Relation (IDREL);

create index FKOrigine
     on Relation (ORIG_IDCON);

create index FKFin
     on Relation (FIN_IDCON);
```

5.7 Etude de l'interface graphique

La réalisation de l'interface graphique consistera à définir son aspect visuel et son utilisabilité en se référant le plus possible à la solution idéale. Il suffira ensuite de convertir le résultat dans DIAGEN.

Comme écrit dans le chapitre *Etat de l'art*, un VQL doit être réalisé en tenant compte de sa représentation visuelle et de la manière dont l'utilisateur construira sa requête. La représentation visuelle de notre requête, indiquée dans le chapitre *Solution idéale*, se définit par ces éléments : des blocs (incluant une zone de texte et une zone de propriété), une connexion (ligne) reliant deux blocs, un titre définissant le bloc et un champ propriété (zone de texte)



La requête imbriquera un titre, un ensemble de propriétés, un ensemble de concepts, un ensemble de relations et un ensemble de liens concepts-relations. Lorsque nous définissons la représentation visuelle d'un concept ou d'une relation, nous constatons qu'elle a les mêmes spécificités que la requête. Petite exception pour la relation qui ne nécessite pas d'ensemble de propriétés. Donc nous pouvons dire que la représentation visuelle d'une requête sera la même que celle d'un concept ou celle d'une relation.

Deux blocs seront reliés par une connexion. Pour que le lien bloc – connexion puisse avoir une signification dans DIAGEN, ces deux éléments doivent obtenir un point commun : celui-ci s'appelle *secteur d'attachement*[35]. Tous les blocs auront un ou plusieurs secteur d'attachement. Pour l'utilisateur ceci se traduit par l'obligation de relier deux blocs en tirant une ligne d'un bloc à l'autre ou plutôt d'un secteur d'attachement à un autre. Dans DIAGEN lorsque deux composants sont reliés par une ligne, celle-ci est surlignée en bleu. Ce procédé intervient après l'analyse du modèle de diagramme, si la connexion

[35] cfr. Présentation de DIAGEN

respecte la syntaxe et la sémantique du diagramme alors la « mise en couleur » est autorisée sinon la ligne reste en noir.

5.8 Présentation de la solution réduite

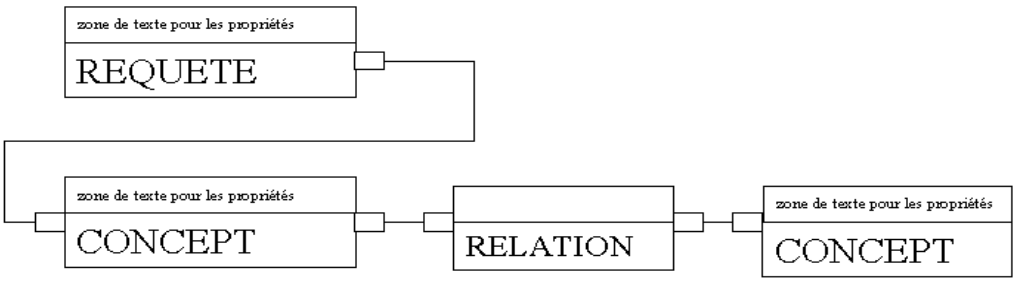
Remarque concernant DIAGEN :

DiaGen n'est pas fourni avec une documentation complète. La seule documentation disponible est l'overview, deux tutoriaux sur les composants et la construction pas à pas d'un exemple basique. Le temps nous a manqué pour une approche très poussée sur DiaGen. L'idéal aurait été de contacter l'université UniBwM en Allemagne pour avoir une documentation complète ou au moins une assistance.

De ce fait nous n'avons pas pu utiliser la présentation visuelle de la solution idéale.

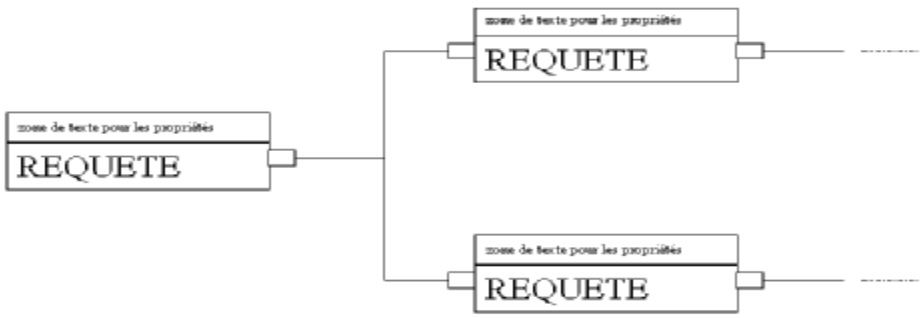
L'interface graphique de la solution réduite sera basée sur l'interconnexion de blocs de type requête, concept et relation. C'est-à-dire que l'utilisateur pour commencer devra poser un bloc de type « Requête » puis le relier à un bloc de type « Concept » Le concept sera relié à une relation qui elle-même sera liée à un autre concept.

Avec ce scénario nous aurons comme résultat, le schéma suivant :



Le schéma ci-dessus a pour seul but de montrer la nouvelle structure d'une requête, aucune signification de sens par rapport à un domaine ne peut être déduite de ce schéma. Celui-ci permet de réaliser une requête de base du type projection par exemple. Pour couvrir les autres caractéristiques techniques de la solution idéale, il faut mettre à disposition de l'utilisateur le moyen de concevoir les requêtes avec jointure[36].

Réaliser ces requêtes sera possible dans un premier temps en ajoutant un secteur d'attachement en entrée (à gauche) du bloc de type « requête »

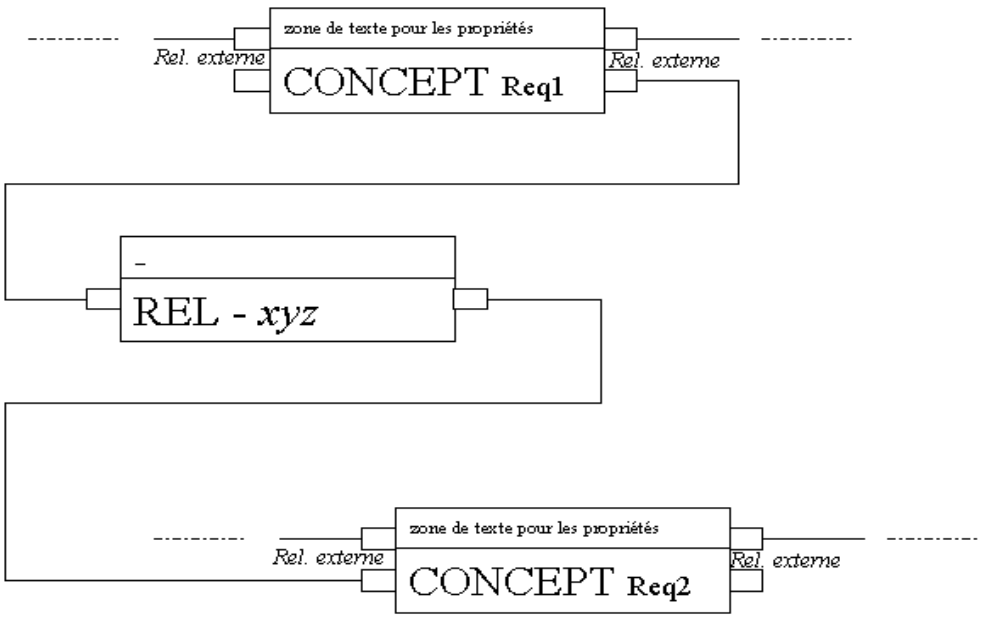


[36] cfr. Chapitre 3.5.5 de la Solution Idéale « Sémantique du langage »

Puis de modifier les concepts en leur donnant la propriété dites « Corrélation externe » Cette propriété change le sens des concepts car ils seront les seuls à pouvoir être liés avec les concepts d'une autre requête.

De ce fait d'un point de vue technique, nous sommes obligés de nous donner la possibilité de distinguer les relations internes à une requête et les relations externes, celles qui unissent des concepts de requêtes différentes.

La solution sera de demander à l'utilisateur de définir les relations externes sur des secteurs d'attachement différents de ceux prévus pour les relations internes. Le schéma ci-dessous nous montre comment relier deux concepts de deux requêtes différentes.



Avec ces deux techniques de réalisation de requêtes nous sommes capables de concevoir les caractéristiques suivantes de la solution idéale :

- Projection
- Jointure
- Sélection
- Union
- Différence d'ensembles
- Intersection
- *Expressions arithmétiques*
- *Agrégats*
- *Filtre sur une chaîne de caractère*

Les trois dernières caractéristiques sont à implémenter dans les propriétés. Donc la solution peut correspondre à une boite de dialogue lors de l'affichage des propriétés d'un concept ou d'une requête.

Il reste dans les différents points de la sémantique d'une requête de la solution idéale l'abstraction et les requêtes imbriquées. Ceux-ci ne seront pas réalisés car nous pensons que les utilisateurs peuvent couvrir quatre-vingt dix pour cent à quatre-vingt quinze pour cent de leurs requêtes avec les moyens mis à leur disposition. Les cinq ou dix pour cent restants peuvent être réalisés avec l'aide du service informatique[37].

[37] cfr. § 5.8.2

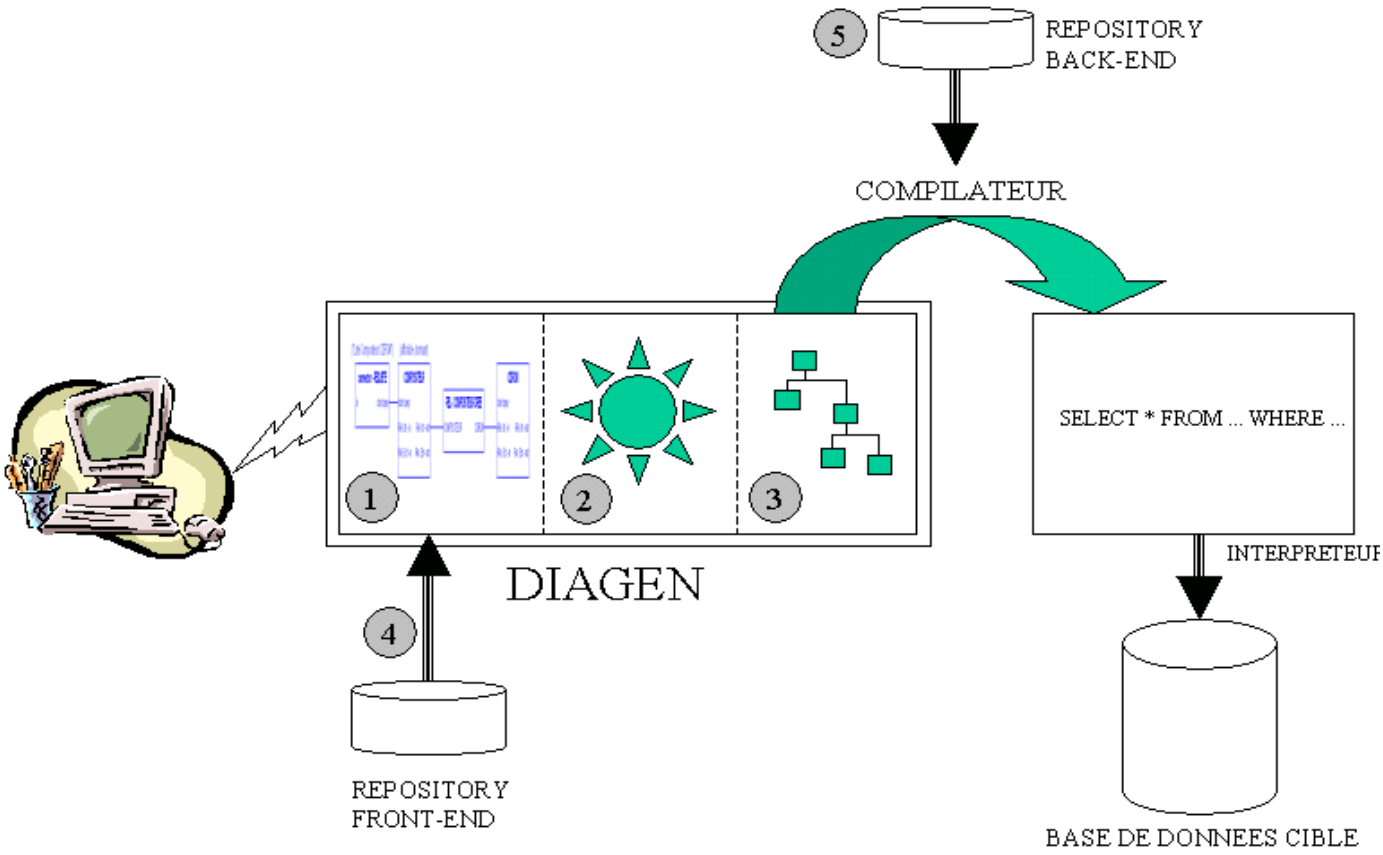
Donc fournir ces deux techniques aux utilisateurs n'apportera pas de valeurs ajoutées au produit. Dans l'analyse des besoins, il est indiqué que les requêtes des utilisateurs admettent une complexité du niveau des jointures uniquement.

L'implémentation de l'interface graphique consiste à développer un outil qui fournit à l'utilisateur la possibilité de créer une question sur base des blocs de types requête, concept et relation. Ces blocs respectent le design défini dans les paragraphes précédents. L'ensemble des blocs assemblés par l'utilisateur respecte aussi la sémantique définie ci-dessus[38]. Celle-ci est insérée dans DIAGEN en définissant la grammaire avec une méthode de l'université UniBwM. En l'incluant dans un fichier appelé « spec », le framework de DIAGEN créera les composants de base pour l'interface graphique. Nous avons développé la représentation[39] appliquée à ceux-ci. Tous les concepts et relations contenues dans le repository sont chargés au lancement de l'outil. Pour cela nous avons réalisé un Data Access Object[40] qui récupère les données résultant des requêtes définies dans le fichier composant/connexion.properties.

Le détail de la définition de la grammaire et de son écriture dans DIAGEN est indiqué dans l'annexe B de ce document.

Le nom donné à l'outil est « **VQLforUser** ».

5.8.1 Aperçu du fonctionnement de « VQLforUser »



[38] Mais aussi celle de la solution idéale
[39] cfr point 1 dans l'aperçu du fonctionnement
[40] appelé aussi DAO : plus d'information à ce lien : http://java.sun.com/blueprints/patterns/DAO.html

Description des modules :

1. Interface graphique
2. Analyseur de diagramme
3. Générateur d'arbre syntaxique
4. Repository du domaine conceptuel de l'utilisateur
5. Repository des liens entre le domaine de l'utilisateur et la base de données cible.

VQLforUser se compose d'un interface graphique (1)[41], d'un modèle de diagramme en mémoire et d'un analyseur de diagramme (2), d'un générateur d'arbre syntaxique(3), d'un compilateur, d'un interpréteur SQL et de deux repository (4)(5).

L'utilisateur créera sa question sur base de sa connaissance du domaine. Pour que l'interface puisse lui fournir tous les éléments nécessaires à la formulation de sa question, nous avons dû ajouter à l'interface de DIAGEN deux nouvelles boîtes de dialogues contenant chacune une liste. La première contient les opérations et concepts disponibles dans le domaine, la deuxième contient la liste des relations possibles pour interconnecter les éléments de la liste précédente. Ces données de type 'concept' et de type 'relation' sont contenues dans le 'repository front-end'(4). Pour que l'interface puisse les récupérer, nous avons réalisé un Data Access Object qui est appelé au lancement de VQLforUser. Nous avons utilisé Microsoft Access comme base de données du Repository. Celle n'est pas imposée, c'est pour cela que le repository peut-être implémenté dans une autre en adaptant le fichier 'bd.properties'[42]. Dans le cas où le SQL de la nouvelle base de données ne serait pas compatible avec le SQL standard, il faudra modifier les requêtes pour interroger le repository. Pour éviter de toucher au code nous les avons rendues accessibles dans le fichier 'connexion.properties'[43].

Tout au long de la formulation de sa question, l'utilisateur est informé si la sémantique utilisée est correcte. DIAGEN prévoit, au moment où un lien est relié à un bloc, qu'un contrôle sémantique soit lancé. Nous avons modifié ce contrôle pour qu'il se base non plus par rapport au type de bloc, comme prévu à la base dans DIAGEN, mais par rapport au sens que nous lui attribuons dans le domaine de l'utilisateur. Cette valeur est contenue dans le repository front-end. Dans le cas d'une erreur, le lien fautif ne sera pas attaché au bloc et sa couleur ne sera pas bleue mais noire. Cette vérification « au fil de l'eau » de la syntaxe et de la sémantique du diagramme est possible grâce à la grammaire définie lors de la configuration de DIAGEN. Le module qui est chargé de cette vérification est l'analyseur de diagramme (2). Celui-ci a en mémoire une représentation de la question de l'utilisateur sous forme de diagramme. Cette mémorisation s'appelle le *modèle de diagramme* dans DIAGEN. Ce modèle va permettre un contrôle en permanence de la syntaxe et de la sémantique de la question de l'utilisateur. Le fichier de grammaire que nous avons réalisé à l'annexe D et expliqué à l'annexe B, est la base de la construction et des contrôles du diagramme. DIAGEN serait sinon incapable d'analyser la question.

Lors de la vérification de la sémantique du modèle, l'analyseur fournit les informations nécessaires au générateur de l'arbre syntaxique (3). L'explication de la construction de celui-ci et la manière dont il est structuré, est expliqué à

[41] numéro de référence sur le schéma ci-dessus.
[42] 'bd.properties' se situe dans le répertoire implementation/src/bd du CD-ROM.
[43] 'connexion.properties' se situe dans le répertoire implementation/src/composant du CD-ROM.

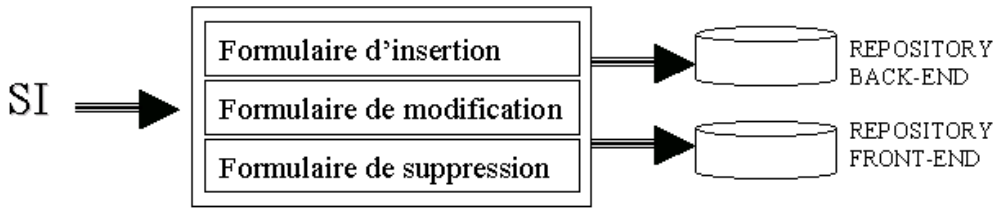
l'annexe C. Nous avons modifié la construction de l'arbre syntaxique prévue dans DIAGEN pour qu'elle corresponde à la grammaire de notre interface. L'arbre syntaxique contient des connexions de blocs. A partir de l'arbre syntaxique nous sommes capables de reformuler une requête SQL. Le rôle de la construction de la requête est donné au compilateur. Il peut sur base de l'arbre et des données du repository back-end construire la requête SQL. Cette étape est expliquée dans le paragraphe 'perspectives futures'. Nous n'avons pas implémenté cette partie par faute de temps. L'implémentation sera du même type que celle réalisée lors des TP's du cours de Syntaxe et Sémantique en première année de Licence.

Avec la requête SQL, l'utilisateur pourra utiliser n'importe quel interpréteur pour interroger la base de données ou même l'envoyer à un de ses collègues.

5.8.2 Les interventions du SI[44]

Comme indiqué dans l'analyse des besoins ou dans l'introduction, le SI est souvent surchargé. VQLforUser diminue-t-il la surcharge du SI ? Améliore-t-il le travail de maintenance du SI ? Cette partie va tenter de vous convaincre que 'oui'.

VQLforUser est basé sur deux repository qui seront maintenus par le SI. Comme indiqué dans le schéma ci-dessous, un outil de maintenance devra être réalisé en support au VQLforUser pour permettre de maintenir les liens, via des formulaires, entre le repository front-end et le repository back-end.



Ces formulaires permettront aux informaticiens du SI d'encoder une seule fois les informations pour écrire dans les deux repository. Ce principe sera applicable pour la modification et la suppression d'informations.

Comme indiqué dans la problématique, le SI était obligé de réaliser des formulaires pour aider les utilisateurs à interroger la base de données. Ici dans le cas de requêtes très complexes, un informaticien a à sa disposition deux méthodes pour aider très facilement un utilisateur. La première est de réaliser sur l'interface de VQLforUser la requête à la place de l'utilisateur et de lui envoyer par mail le fichier de la question. L'utilisateur dans ce cas n'aura plus qu'à modifier les paramètres pour la maintenir. Cette méthode sera utilisée dans le cas de questions bien spécifiques à un utilisateur. La deuxième vise plus à étendre à tous les utilisateurs l'accès à la question. L'informaticien à l'aide de l'outil de maintenance encodera la requête SQL correspondant à la question sous un concept. Tous les utilisateurs, dans ce cas, pourront l'insérer dans d'autres requêtes. Pour comprendre ce principe nous vous invitons à lire le paragraphe 'Perspectives futures'.

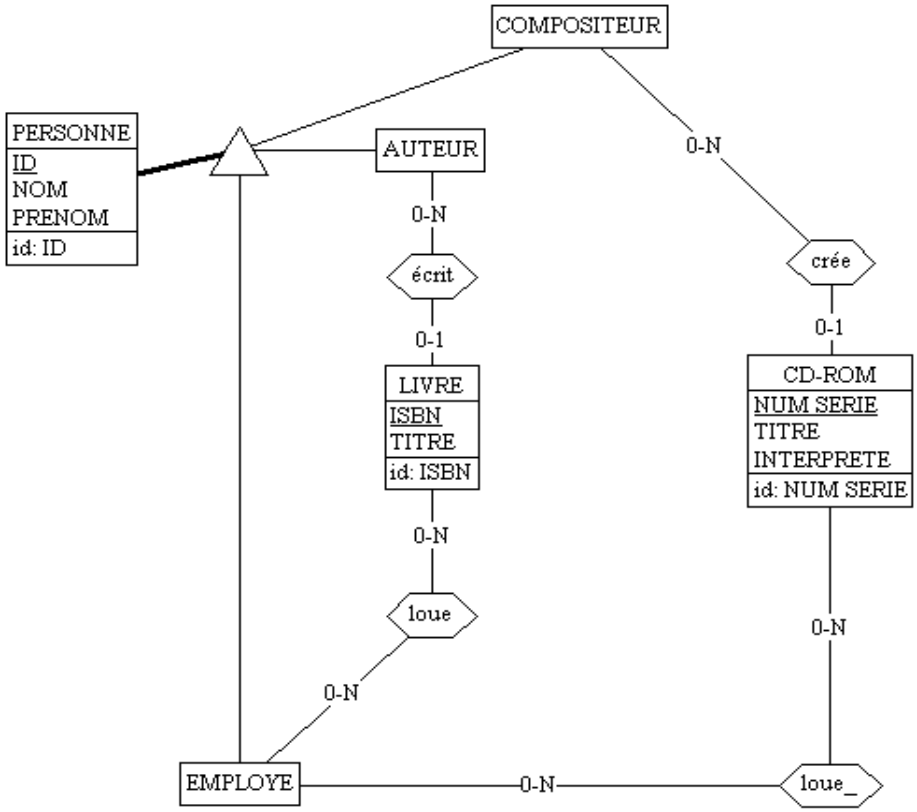
Grâce à l'outil de maintenance la gestion des deux repository est facilitée et surtout plus conviviale pour les informaticiens du SI. Ils n'auront plus à

[44] SI : service Système d'information (ou encore appelé service Service Informatique) dans une société.

développer ou à étendre des applications pour rendre service aux utilisateurs. Avec VQLforUser, ils pourront très rapidement répondre aux besoins des utilisateurs, tout en sachant que déjà avec VQLforUser dans les mains des utilisateurs, les demandes auprès du SI diminueront très fortement. L'utilisateur est déjà autonome pour interroger la base de données avec des questions complexes.

5.8.3 Exemple complet d'utilisation

Si nous considérons ce domaine de départ :



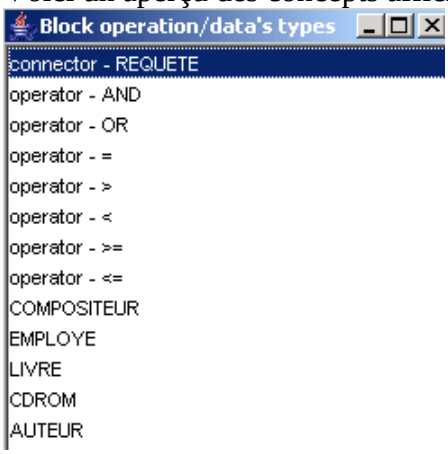
L'écriture dans le repository consistera à encoder tous les concepts et toutes les relations de ce schéma dans les tables adéquates. C'est-à-dire sur ce schéma nous identifions les concepts : Personne, Compositeur, Auteur, Livre, CD-ROM et Employé. Nous écrirons dans la table CONCEPT, les données ci-dessous :

	ID	TYPE	NOM	DESCRIPTION
	1	compositeur	COMPOSITEUR	COMPOSITEUR
	2	employe	EMPLOYE	EMPLOYE
	3	livre	LIVRE	LIVRE
	4	cd	CDROM	CDROM
	5	auteur	AUTEUR	AUTEUR

Le concept PERSONNE n'est pas encodé car il n'apporte rien à la compréhension du domaine. Ici l'exemple permet de montrer que le service SI ne doit jamais perdre de vue la pertinence des concepts qu'il encode. Il ne doit jamais oublier l'utilisateur qui va manipuler ces concepts.

Dans la table CONCEPT du REPOSITORY nous avons ajouté le champ Type[45]. L'intérêt se justifie dans la sémantique du diagramme. Ce champ va permettre de corriger l'utilisateur s'il décide de relier par exemple un concept « Auteur » au secteur d'attachement « Compositeur » de la relation « compositeur crée ». Les types étant différents la connexion est refusée. L'utilisateur s'apercevra de son erreur puisque la connexion sera affichée en noire au lieu d'être en bleu.

Voici un aperçu des concepts affichés dans la liste de DIAGEN :



Sur le schéma conceptuel, nous identifions les relations : Auteur écrit des livres, employé loue des livres, compositeur crée des cd-rom, employé loue des cd-rom. Ce qui donne comme relations encodées dans la table RELATION :

	ID	ID_ORIG	ID_FIN	DESCRIPTION
	1	5	3	AUTEUR ECRIT
	2	2	3	LOUE LIVRE
	3	2	4	LOUE CDROM
	4	1	4	COMPOSITEUR CREE

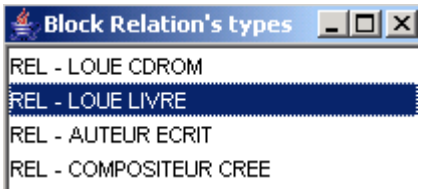
Nous avons raccourci les descriptions des relations pour éviter du surcharger la liste. L'utilisateur ne perdra pas pour autant de l'information puisque après avoir posé le bloc il y verra les concepts d'origine et de fin affichés à l'intérieur. Pour exemple voici la relation « loue livre ». On y voit bien le concept d'origine auquel devra être relié un concept « employé »[46] et le concept de fin auquel devra être relié le concept « livre »[47] :

Voici un aperçu des relations affichées dans la liste de DIAGEN :

[45] Les valeurs du champ Type sont limitées au contenu du fichier 'Types.properties'. Celui-ci est situé dans le package « composant ». (cfr. Annexe A / Répertoire 'src' dans le CD ROM)

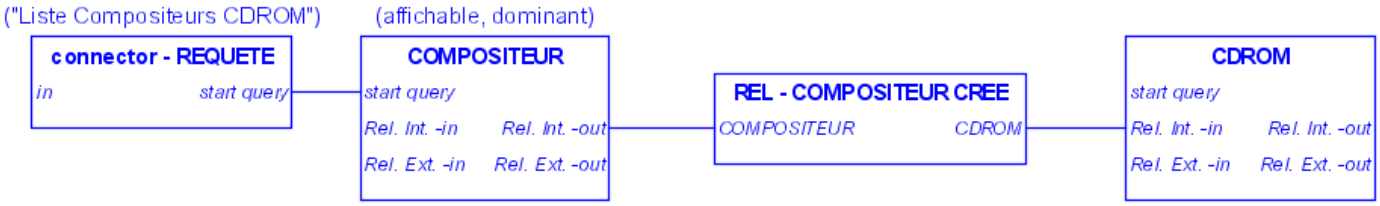
[46] ou un dérivé du concept « employé ». Par exemple cela peut être un concept « employé chef d'équipe ».

[47] ou un dérivé du concept « livre ». Par exemple cela peut être un concept « livre d'aventure ».

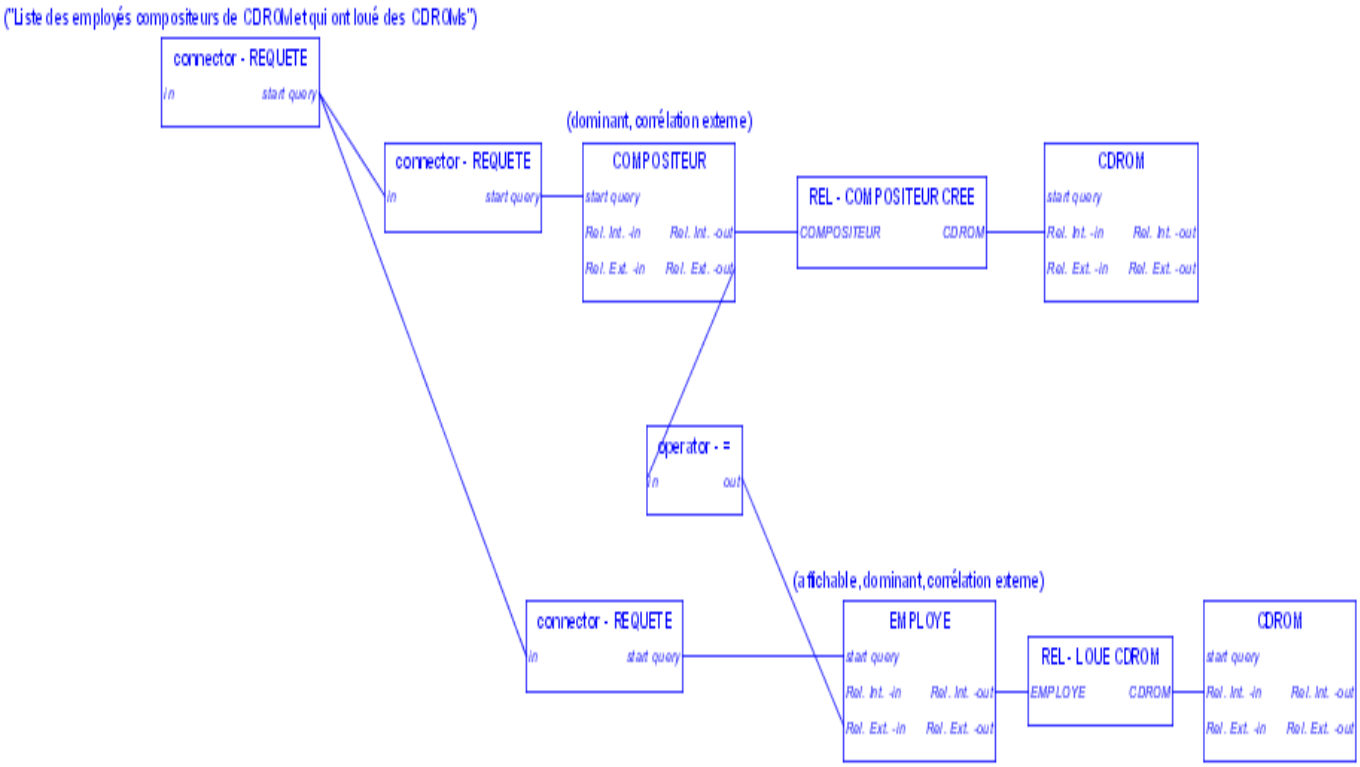


Exemple de requêtes

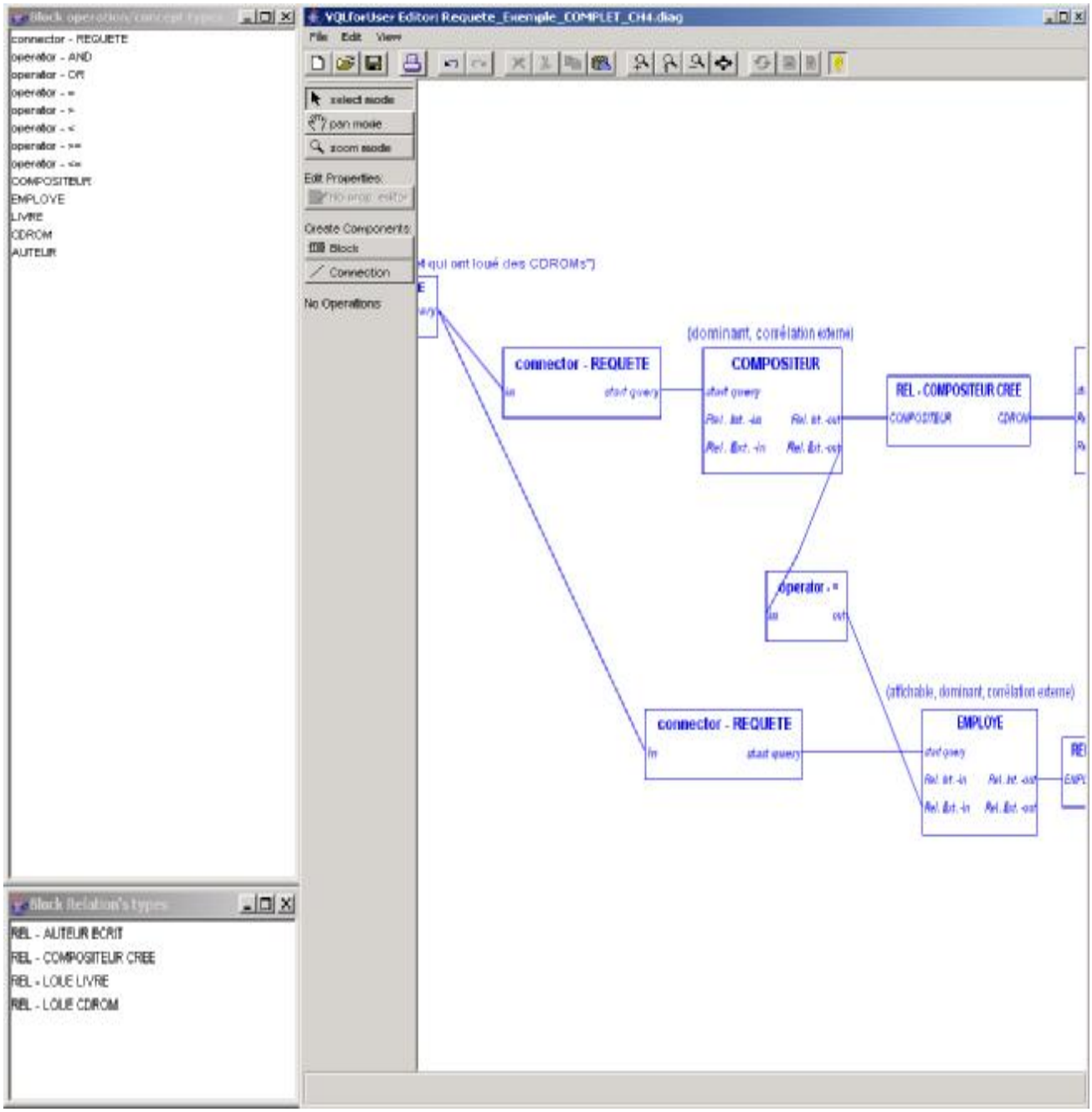
Requête : *Liste des compositeurs qui ont créé un CDROM*



Requête : *Liste des employés compositeurs de CDROM et qui ont loué des CDROM*



Voici une vue complète de l'éditeur :



5.8.4 Perspectives futures

Nous n'avons pas implémenté le compilateur en relation avec le repository back-end. D'où le but de ce paragraphe qui montrera comment nous avons pensé cette partie si nous avions dû la réaliser.

Principe

L'arbre syntaxique sera traité par d'autres composants tel que le convertisseur défini dans l'architecture conceptuelle. Le principe est de définir dans le convertisseur la correspondance entre le schéma conceptuel et le schéma physique. Pour cela nous avons besoin d'un élément qui contiendra les définitions de tous les concepts qui sont implémentés dans la base de données et il devra indiquer comment les relations sont traduites du schéma conceptuel vers le schéma physique. La solution est de mettre en relation un repository back-end avec le convertisseur dont l'objectif sera de contenir les liens entre le domaine de l'utilisateur, décrit dans le repository front-end et la base de données cible.

Définition du contenu du repository

Nous avons pensé à cette définition pour chacun des éléments définis dans le repository front-end :

- Type :
  pour un concept
  'Type' définit le cas où la propriété du concept est indiquée comme 'dominant' alors cette définition est valable.
  pour une relation
  'Type' indique où interviendra la relation dans la requête SQL.

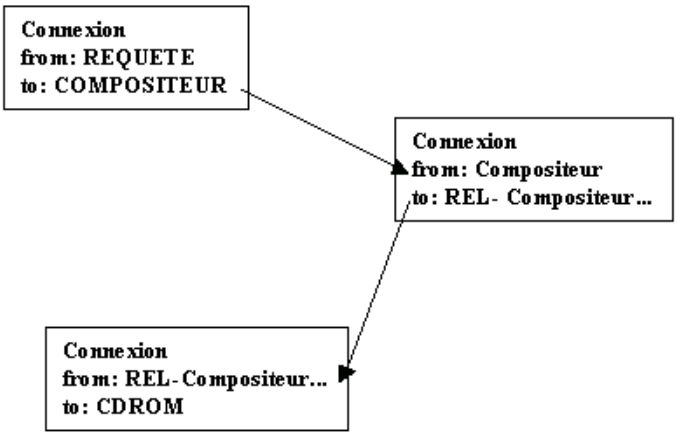
- Classe :
  pour un concept
  'Classe' indique si dans la base de données cible le concept est une table, un champ ou une requête.
  pour une relation
  'Classe' indique si dans la base de données cible cette relation est une table, un champ, une jointure de type 'where' ou de type 'join'

- Value :
  'Value' indique la valeur correspondante à la classe.

Fonctionnement du compilateur

Le compilateur lira l'arbre syntaxique pour l'ordonner et le découper[48] (dans le cas de relations externes). Prenons l'arbre syntaxique du cas simple de notre exemple ci-dessus :

[48] Pour plus d'explications voir l'annexe C



L'élément racine d'une requête sera toujours la connexion ayant comme valeur 'REQUETE' dans l'attribut 'from'. Ensuite l'arbre sera parcouru à travers les différents attributs 'to' et 'from' de chaque connexion rencontrée. Une connexion fournit des informations sur un bloc et une relation. Etant donné que les connexions mémorisent les blocs suivants et les blocs précédents, leurs informations sont accessibles à deux endroits de l'arbre et plus précisément sur deux connexions. Nous pouvons prendre comme règle de départ de traiter le bloc de l'attribut 'to' de chaque connexion.

Construction de la requête

Nous définissons que chaque requête sera encapsulée par :

select * from (*[..1..]* **)** **where** *[..2..]*

'*[..1..]***'** sera remplacé par les concepts trouvés sur chaque connexion rencontrée dans l'arbre et dont les valeurs correspondront au contenu du repository back-end .

Idem pour '*[..2..]*' mais pour les relations.

Pour une requête simple, nous obtenons un arbre de ce type :

Connexion : from -> REQUETE
to -> CONCEPT1

Connexion : from -> CONCEPT1
to -> RELATION1

Connexion : from -> RELATION1
to -> CONCEPT2

Dans ce cas le compilateur donnera comme requête :

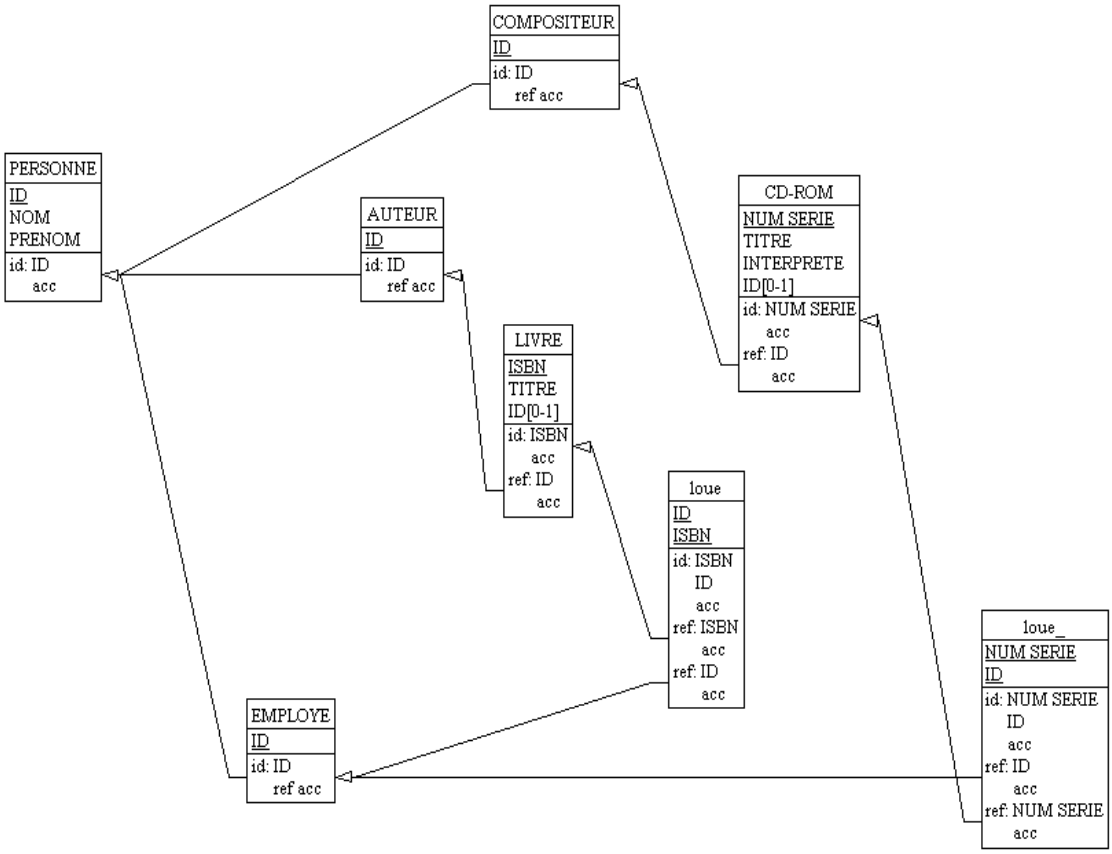
select * from (
*valeur_CONCEPT1***,**
valeur_CONCEPT2
) **where**
valeur_RELATION1

Avec ce principe nous pensons que les requêtes simples peuvent être générées. Mais ceci n'est pas valable pour les cas plus complexes par exemple quant, au niveau conceptuel, une relation est définie entre deux entités mais, au

niveau physique, celle-ci a été transformée en une table. Dans ce cas, la relation doit être définie dans le 'from' et dans le 'where'. Pour résoudre ce problème, il faut gérer des 'types' de relations qui permettront de savoir où placer les valeurs des relations dans la requête.

Exemple cas concret

Voici le schéma logique de la base de données lié au schéma conceptuel de l'exemple du paragraphe précédent :



La base de données pourrait être décrite comme ci-dessous dans le repository back-end :

Concept COMPOSITEUR :
- Type : ID
- Classe : SELECT
- Value : *select * from personne p, compositeur c where p.id=c.id*

Concept PERSONNE :
- Type : ID
- Classe : Table
- Value : PERSONNE

Concept CDROM :
- Type : ID
- Classe : Table
- Value : CDROM

Concept EMPLOYE :
- Type : ID
- Classe : SELECT

Value : *select * from personne p, employe e where p.id=e.id*

… Ainsi de suite pour toutes les tables.

Relation « COMPOSITEUR CREE » :

Type : WHERE
Classe : Jointure
Value :
[out].[FK_CDROM_COMPOSITEUR] = [in].[ID_COMPOSITEUR]

Relation « EMPLOYE LOUE CDROM » :

Type : WHERE_FROM
Classe : Jointure
Value WHERE :
[in].[EMPLOYE_ID] =[TABLE_LOUE_].[FK_LOUE__EMPLOYE_ID]
AND [TABLE_LOUE_].[FK_LOUE__CDROM_ID] = [out].[CDROM_ID]
Value FROM :
[TABLE_LOUE_]

… Ainsi de suite avec les relations.

Une relation relie deux concepts. Concepts que nous ne pouvons connaître à l'avance. C'est pour cela que la valeur de la relation « compositeur crée » est :
[out].[FK_CDROM_COMPOSITEUR] = [in].[ID_COMPOSITEUR]

Où, *[out]* correspond à l'alias défini dans la clause 'from' pour le concept en output de la relation. Idem pour *[in]* mais pour le concept en input de la relation. *[FK_CDROM_COMPOSITEUR]* correspond à la valeur associée au champ de la clé étrangère de la table *[out]*. Idem pour *[in]*. Toutes ces valeurs seront contenues dans une table de références du repository back-end.

L'exemple simple « Liste des compositeurs qui ont créé un CD-ROM » peut être réalisé comme suit :

Traduction de l'arbre syntaxique :

```
select * from
(select * from personne p, compositeur c where p.id=c.id) comp,
cdrom cd
where [out].[FK_CDROM_COMPOSITEUR] = [in].[ID_COMPOSITEUR]
```

Après récupération des valeurs contenues dans une table de référence, la requête sera :

```
select * from
(select * from personne p, compositeur c where p.id=c.id) comp,
cdrom cd
where cd.COMPOSITEUR_id = comp.id
```

Pour réaliser l'exemple « Liste des employés compositeurs de CDROM et qui ont loué des CDROM », nous créons dans un premier temps la requête « Liste des employés qui ont loué un CDROM ». Le résultat est :

Traduction de l'arbre syntaxique :

```
select * from
(select * from personne p, employe e where p.id=e.id) comp,
cdrom cd,
loue_ l
```

```
where [in].[EMPLOYE_ID] [TABLE_LOUE_].[FK_LOUE__EMPLOYE_ID]
AND [TABLE_LOUE_].[FK_LOUE__CDROM_ID] = [out].[CDROM_ID]
```

Après récupération des valeurs contenues dans une table de références, la requête sera :

```
select * from
(select * from personne p, employe e where p.id=e.id) emp,
cdrom cd,
loue_ l
where emp.id = l.EMPLOYE_id
AND l.CDROM_NUM_SERIE = cd.NUM_SERIE
```

Pour réaliser la jointure indiquée dans l'exemple, il faut encapsuler les deux requêtes d'un FROM, mettre en amont un select * et ajouter l'opération après un WHERE comme indiqué ci-dessous :

```
select * from
(
select * from
(select * from personne p, compositeur c
where p.id=c.id) comp, cdrom cd
where cd.COMPOSITEUR_id = comp.id) comp_cree_cd,

(select * from (select * from personne p, employe e
where p.id=e.id) comp, cdrom cd, loue_ l
where comp.id = l.EMPLOYE_id
AND l.CDROM_NUM_SERIE = cd.NUM_SERIE) emp_loue_cd
)
where comp_cree_cd.ID = emp_loue_cd.id
```

Cette proposition de solution peut être améliorer en gérant les jointures par des 'join'.

Remarque :
Les codes SQL ci-dessus ont été exécutés sur le SGBD MYSQL pour la vérification de leurs bonnes formes.

6 .Conclusion

L'origine de ce mémoire est une demande concrète, des utilisateurs del'aciérie de CARSID, de simplification de l'interrogation de la base de données Archivage. Il n'est en effet pas facile pour des utilisateurs dits non-informaticien de comprendre l'agencement des tables, d'effectuer des requêtes SQL quand leurs métiers consistent, par exemple, à analyser des processus mécaniques ou d'optimiser des régulations de débit. Les sociétés pourtant aujourd'hui demandent à ces utilisateurs soit de connaître le SQL soit d'attendre la diminution de charge d'un service informatique pour qu'il réalise un formulaire adéquat.

L'objectif de ce mémoire est d'apporter une réponse à cette problématique. La solution est tirée de plusieurs articles scientifiques traitant de cette même problématique et qui apportent comme solution : le Visual Query Language. Il n'est pas évident de découvrir le langage le mieux adapté dans ce monde de VQL. Ils sont si nombreux que percevoir celui le mieux adapté à notre problème est difficile. C'est pour cela que notre « Etat de l'art », basé sur l'article de CATARCI ([CATARCI 95]), réalise une taxonomie des langages visuels d'interrogation de base de données. C'est sur cette base que la pertinence de notre VQL est démontrée.

Questionner, avec les termes de son domaine d'expertise, une base de données, sans se soucier de sa structure et de ses tables, est l'objectif à atteindre pour notre langage.

Dans notre langage, une question s'exprime en terme de concepts et de relations. Mais ce sont des concepts et des relations du domaine de l'utilisateur. Donc la question est construite avec le vocabulaire du modèle conceptuel de la base de données cible.

La solution idéale est décrite en ne tenant pas compte de la réalité des contraintes de réalisation (temps, coût, impact, nécessité?), ainsi nous expliquons plus facilement, autant au niveau syntaxique qu'au niveau sémantique, la consistance de notre langage. Pour cela à partir d'exemples de cas réalistes extraits avec notre langage (de manière théorique), nous montrons l'équivalence avec le SQL.

Mais restons réaliste !

La solution réduite de ce mémoire l’est justement. En prenant tous les besoins des utilisateurs, en définissant leurs profils, en tenant compte des requêtes déjà réalisées, nous sommes certains de couvrir tous les besoins. Nous avons décidé de suivre la méthodologie vue aux travaux pratiques de Méthodologie de Développement de Logiciels pour implémenter notre langage.

L’outil décrit dans ce mémoire, et réalisé en partie, s’appelle VQLforUser. Il répond aux besoins de l’utilisateur par sa simplicité d’utilisation et l’aide qu’il procure en indiquant les erreurs de sémantique. Il permet de construire des requêtes SQL sur base d’une question formulée avec des termes du domaine d’expertise de l’utilisateur. Son fonctionnement : construire en entrée, un diagramme représentant une question, fournir en sortie la requête SQL correspondant. Il suffit ensuite de copier ce code dans un interpréteur pour extraire les données de la BD. Nous avons décider de ne pas imposer un interpréteur ou un outil d’extraction pour laisser une plus grande liberté aux utilisateurs. L’outil offre par ailleurs la possibilité aussi de zoom, d’impression et de sauvegarde des questions.

Lors de la réalisation de VQLforUser, l’architecture concrète a abouti au choix technique de DIAGEN (diagram generator). Cet outil-framework nous permet de réaliser des diagrammes sur base d’une grammaire orientée graphique et d’un framework pour la réalisation des composants graphiques. Nous avons

définit la grammaire à l'aide de la méthode de DIAGEN sur base des SRHG[49] et HGM[50].

Sur base de ce qui précède, nous constatons que VQLforUser répond bien aux besoins de l'utilisateur pour l'interrogation d'une base de données, ce qui était l'objet de notre travail.

Une amélioration de l'outil est envisageable dans les domaines suivant :

- Au niveau de l'utilisateur : convivialité et ergonomie.
- Au niveau des performances : utilisabilité et sécurité.

En ce qui concerne la convivialité et l'ergonomie, nous pourrions permettre à un utilisateur de construire lui-même ces propres concepts sur base d'une question exprimée avec VQLforUser. Il serait intéressant aussi de lui offrir la possibilité de filtrer les champs des tables car actuellement nous retournons tous les champs des tables d'une question.

Pour l'utilisabilité, nous pourrions prévoir une interrogation sur plusieurs bases de données. Ne pas le prévoir est une limitation très contraignante dans les Datawarehouse ou Datamining. Pour pouvoir toucher ces domaines, une solution devra être apportée au niveau du compilateur de notre outil pour que son repository back-end tienne compte de plusieurs bases de données ou alors dans le cas d'un Datawarehouse ou datamining directement extraire les requêtes dans un méta-langage.

Enfin au niveau de la sécurité, certaines données peuvent être sensibles et donc doivent être non visible pour certains utilisateurs, là aussi ne faudrait-il pas filtrer en ajoutant des droits d'accès sur les tables ou alors sur les données. Dans ce dernier cas il faudra prévoir une table de références fournissant les autorisations aux données d'un utilisateur.

[49] Spatial Relationship Hypergrah
[50] HyperGraph Model

Bibliographie

1 [ANGEL90] M. Angelaccio, T. Catarci, G. Santucci (1990) *QBD* : A fully Visual Query System* Journal on Visual Languages and Computing, 1, 2;, 255-273

2 [AUDDINO91] A. Auddino, E. Amiel, B. Bhargava *Experiences with SUPER, a database Visual Environment* in Proc. Of the 2nd Int'l Conf. On Database and Expert Systems Applications – DEXA '91, pp. 172-178, Berlin, 1991

3 [BALKIR97] N.H. Balkir, E. Sukan, G. Ozsoyoglu, Z. M. Ozsoyoglu *VISUAL A Graphical Icon-Based Quey Language*, 1997

4 [BATINI91] Batini C., T. Catarci, M.F. Costabile and S. Levialdi. *Visual Query Systems.* Technical Report N°04.91. Dipartimento di Informatica e Sistemisca, Universita' di Roma "La Sapienza », 1991

5 [CATARCI90] Angelaccio M., T. Catarci and G. Santucci *QBD*: A Graphical Query Language with Recursion.* IEEE Transactions on Software Engineering 16(10); 1150-1163, 1990

6 [CATARCI93] Catarci T. and G. Santucci *Fundamental Graphical Primitives for Visual Query Languages* Information Systems, 1(1), 1976

7 [CATARCI93] Tiziana Catarci, Giuseppe Santucci, Michele Angelaccio, *Fundamental Graphical Primitives for Visual Query Languages* (1993)

8 [CATARCI95] Tiziana Catarci, Maria F. Costabile, Stefano Levialdi, Carlo Batini *Visual Query Systems for Databases: A Survey* 1995

9 [JONES97] Jones, S. and McInnes *A Graphical User Interface for boolean Query Specification.,* Working Paper 97/31 Department of Computer Science, University of Waikato, Hamilton, New Zeland 1997. *Submitted to International Journal on Digital Libraries*

10 [MASSARI95] Antonio Massri (Roma, Italy), Panos K. Chrysanthis (Pittsburgh U.S.A.), *Visual Query of completely Encapsulated Objects* appeared in the Proceedings of the Fifth International Workshop on Research Issues on Data Engineering: Distributed Object Management (RIDE-DOM) pp 18-25, March 1995

11 [MURRAY98] Norman Murray, Norman Paton and Carole Goble *Kaleidoquery: A Visual Query Language for Object Databases* (1998) Department of Computer Science University of Manchester Oxford Road, Manchester, M13 9PL, UK

12 [OLSTON98] Chris Olston, Michael Stonebraker, Alexander Aiken, Joseph M. Hellerstein *VIQING: Visual Interactive QueryING* Department of Electrical Engineering and Computer Sciences University of California at Berkeley

13 [SANTUCCI94] Tiziana Catarci, Giuseppe Santucci *Query By Diagram : A graphical Environment For Querying Databases* Dipartimento di informatica e Sistemistica, Universita' degli Studi di Roma "La sapienza", 1994

14 [DIAGEN] Site: http://www2-data.informatik.unibw-muenchen.de/DiaGen/
Version utilisée pour ce mémoire : 2.1a

ANNEXES

Annexe A : Contenu du CD-ROM

Le CD-ROM joint à notre mémoire contient les répertoires suivants:

/memoire/	contenant ce mémoire aux formats Microsoft Word (DOC)et Adobe Portable Document Format (PDF).
/rational/	contenant tous les diagrammes de ce mémoire réalisés avec Rational Rose.
/dbmain/	contenant tous les schémas réalisés avec DBMain lors de la rédaction de ce mémoire.
/interviews/lib/	contenant le logiciel istar
/interviews/doc/	contenant les .doc des interviews et l'arbre des buts réalisé avec istar.
/demo/	contenant 3 vidéos de démonstration du prototype dans 3 cas de figure : 1-sample_success.avi (1min19sec) 2-sample_not_success.avi (2min54sec) 3-sample_complete.avi (5min22sec)

/implementation/bin/ contenant les fichiers compilés (.class) de notre outil.

/implementation/src/ contenant les fichiers sources (java, properties) de l'outil.

/references/ contenant les articles repris dans ce mémoire, plus d'autres intéressants.

Annexe B : Implémentation de l'interface graphique

L'interface graphique est l'éditeur qui permettra de construire un arbre qui traduira la requête dessinée par l'utilisateur en structure formelle compréhensible par un compilateur.

L'éditeur de DIAGEN fonctionne avec le modèle de composants qui est lui-même dépendant de la syntaxe formelle. Configurer un éditeur dans DIAGEN signifie de décrire quatre aspects différents :

- L'aspect visuel du diagramme c'est-à-dire les composants visibles du diagramme et les relations spatiales entre eux qui sont les plus importants.
- La structure du diagramme logique qui est décrite par des règles de transformation d'hypergraphe et d'une grammaire d'hypergraphe.
- Des contraintes sur la disposition du diagramme qui aident à maintenir la structure.
- Des opérations de syntaxe-orientée arborescence (semblables aux macros) qui fournissent une manière facile de mettre en application les manipulations complexes du diagramme qui sont fréquemment nécessaire. Ceci ne sera pas utilisé dans notre outil.

DIAGEN suggère que nous développions les spécifications d'un type de diagramme dans l'ordre suivant :

1. Définir la syntaxe formelle pour les diagrammes (la structure du SRHG) : ceci correspond à déterminer de quels composants nous aurons besoin et comment ils seront reliés
2. Ecrire les classes Java qui représentent les composants de diagramme et coder les attributs spatiaux des relations.
3. Indiquer les transformations réductrices et la grammaire d'hypergraphe pour analyser une représentation de SRHG du diagramme.
   Cette étape est en grande partie indépendante de l'étape 2, mais l'inscription du code Java nous permet d'abord de créer un éditeur exécutable de test pour essayer notre grammaire
4. Définir les contraintes appropriées pour les relations structurales reconnues qui sont préservées quand le diagramme est modifié.
5. Définir des opérations d'édition complexes comme transformations du SRHG

Dans ce chapitre nous nous attarderons sur l'aspect de la grammaire et de sa construction. Nous ne décrirons pas la manière dont nous avons codé en Java les aspects graphiques : suffisamment de documentations dans les librairies existent pour aider le lecteur dans ce domaine. De ce fait le point 2 ne sera pas décrit.

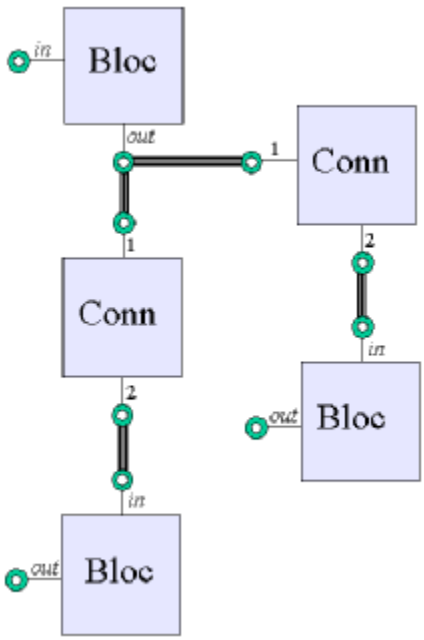
SRHG (syntaxe formelle de notre diagramme)

La technique principale qui sera mise à la disposition d'un utilisateur pour écrire sa requête est la connexion de concepts et de relations. Nous avons défini que les concepts, requêtes et relations étaient au niveau visuel des blocs, appelé ci-

dessous ‘Bloc’. Nous considérons aussi que le trait qui unit deux blocs est aussi un élément visuel que nous nommerons dans le schéma ci-dessous ‘Conn’.

Nous écrirons la grammaire dans un fichier ayant une structure spécifique au compilateur du framework de DIAGEN. Le contenu de ce fichier sera décrit au fur et à mesure de l’analyse.

Le schéma ci-dessous représente l’hypergraphe de notre diagramme (SRHG) Le but de l’hypergraphe est de modéliser les liens entre les éléments graphiques pour en définir une grammaire.



Nous déduisons que le diagramme se base sur un bloc de départ puis s’étend en y ajoutant à chaque fois un élément ‘Conn’ et un élément ‘bloc’. Ce dernier est donc un élément *non-terminal* dans la grammaire. Attention le bloc de départ est l’exception puisqu’il sera considéré comme étant un élément *terminal*. On peut donc écrire ces quelques lignes dans le fichier de grammaire :

nonterminal Chart[0] { Chart chart; },
Block[1] { Block block; Chart chart; };

Nous avons ajouté dans la liste des non-terminaux : le ‘chart’. Il correspond à l’environnement de départ. Celui-ci permettra de transmettre l’arbre syntaxique[51] et de maintenir un lien avec le pointeur sur l’arbre.

Entre ‘{}’ nous écrivons une partie du code Java qui sera ajouté par le compilateur du framework. Nous initialisons les composants graphiques qui seront utilisés dans le code de la grammaire.

terminal fb[1] { Block block; },
to[2] { Connector fromConn, toConn; Connection conn; },

fb pour “first bloc”: Il correspond au premier bloc posé par l’utilisateur.

to signifie le reste (‘vers’) : Il contiendra les connecteurs précédent et suivant, ainsi que toutes les caractéristiques des deux blocs et de la connexion : celles-ci seront mémorisées dans la classe ‘Connection’.

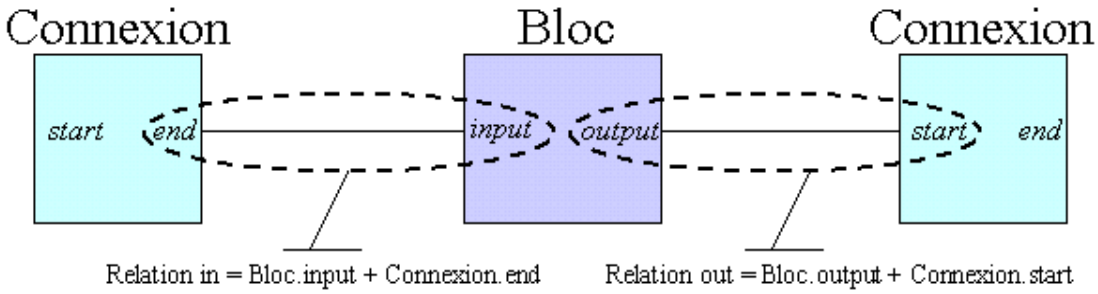
[51] Pour rappel l’arbre syntaxique est construit au fur et à mesure de la construction du diagramme.

Une connexion possède deux secteurs d'attachement[52]. Le connecteur 1 sera le début de la connexion, le 2 sera celui de fin. Un bloc a au moins deux connecteurs[53] : *input* et *output*. Sur le schéma nous nous apercevons qu'à la sortie d'un bloc nous pouvons *attacher* plusieurs connexions.

Ces deux lignes dans le fichier de grammaire décrivent les réflexions ci-dessus :

```
component block[*] { Block{input, output} },
          conn[2] { Connection[Start,End] };
```

Nous pouvons ajouter qu'un composant 'bloc' aura toujours plusieurs *paire* 'input' et 'output'de connecteurs et qu'une connexion aura toujours deux connecteurs 'Start' et 'End'.



Afin d'obtenir cette représentation visuelle et de permettre le lien graphique entre une connexion et un bloc, nous ajouterons ces quelques lignes :

```
relation in[2]  { Relations.intersect(input,End) },
          out[2] { Relations.intersect(output,Start) };
```

La structure de la grammaire est définie comme suit:

```
grammar {
  start Chart<chart>;

  ch:Chart() ::= << b:Block(_) >> {
                ch.chart = Semantics.createChart(<<b.block>>);
                b.chart = ch.chart;
            };

  bl:Block(a) ::= b:fb(a) {
                bl.block = b.block;
                };

  embed t:to(a,b) into bl1:Block(a) * bl2:Block(b) {
                    Semantics.addConn(bl1.chart,
                                      t.conn,
                                      bl1.block, t.fromConn,
                                      bl2.block, t.toConn);
                };
}
```

[52] appelé par la suite « connecteur »
[53] Ici nous ne représentons pas les aspects de relations internes et externes.

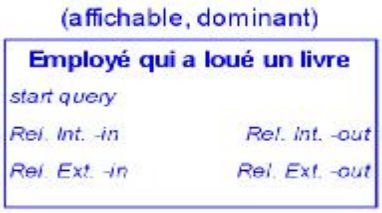
Ces lignes signifient que le 'chart' contient plusieurs blocs. Ils seront pris un par un et considérés comme étant un *fb*. Dans ce cas chaque relation avec une connexion sera analysée et si elle respecte la règle[54] qu'un bloc 'a' a une relation avec un bloc 'b' alors on l'ajoute dans l'arbre syntaxique.

Représentation graphique

La représentation visuelle a défini deux formes d'éléments graphiques :

- Le bloc : il représente un concept ou une relation
- La connexion : elle met en relation deux blocs.

Un bloc représentant un concept sera affiché comme suit :



Le bloc 'concept' : « Employé qui a loué un livre » contient 3 connecteurs en entrée et 2 connecteurs en sortie. Ceux en entrée sont :

- *start query* qui est le connecteur utilisé dans le cas où ce bloc serait le premier bloc 'concept' posé de la requête.
- *Rel. Int.-in* qui signifie « relation interne en entrée ». Sur ce connecteur sera fixé une connexion ayant à l'autre extrémité un bloc 'relation'.
- *Rel. Ext.-in* qui signifie « relation externe en entrée ». Elle est le pendant à la relation interne.

En sortie nous n'avons que deux connecteurs. Ils sont les autres extrémités des connecteurs en entrée. Quand le connecteur Rel. Int-in est utilisé alors le connecteur « Rel. Int.-out » devra l'être à son tour dans l'hypothèse où la requête devait être continuée.

L'intitulé du concept sera indiqué en gras sur la partie du haut du bloc.

Sur le dessus du bloc sera précisé les propriétés liées au concept comme indiqué dans la solution idéale.

Le bloc 'relation' se représente comme suit :

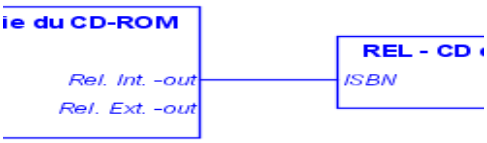
Le bloc 'relation' « REL-employé loue livre » possède deux connecteurs. L'un intitulé « *ID EMPLOYE* » en entrée indique que le type du concept à connecter devra être du type identifiant d'un employé. L'autre intitulé « ISBN » devra être connecter à un concept du type ISBN[55].

L'intitulé de la relation sera indiqué en gras sur la partie du haut du bloc.

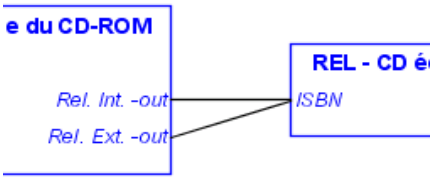
[54] Ecrit *embed* dans la grammaire
[55] Identifiant d'un livre

La connexion sera représentée par un trait entre deux connecteurs. Ce trait sera dessiné au moyen de la souris en cliquant sur un connecteur pour le désigner comme origine puis en cliquant sur un autre connecteur pour le désigner comme cible. Si la connexion est correcte d'un point de vue sémantique[56] alors un trait bleu sera dessiné sinon il sera noir.

Voici un exemple de connexion correcte. Celle-ci est dessinée en bleu :



Ici la connexion supplémentaire entre le connecteur « relation externe-out » et le connecteur de la relation ISBN met en échec d'un point de vue sémantique les deux connexions :



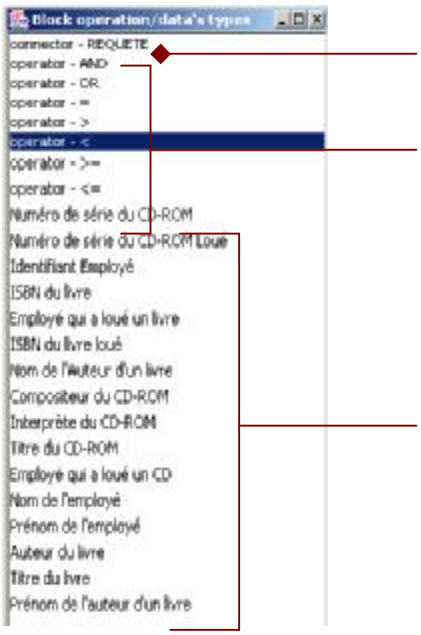
Ici les deux connexions sont noires car nous n'avons pas le droit de joindre deux connexions sur un connecteur en entrée.

Avant de poser des blocs l'utilisateur devra tout d'abord sélectionner le type d'action qu'il va effectuer. C'est-à-dire préciser s'il va poser des blocs ou alors les relier par une connexion. Il le signale en sélectionnant l'un de ces deux boutons :

Après avoir sélectionner le bouton « Block » il doit choisir soit un concept ou soit une relation. Pour cela deux listes sont à sa disposition : une liste de concepts et une liste de relations. Le bloc désigné précisera le type.

[56] cfr. Paragraphe ci-dessous « Réduction de la grammaire »

Liste des concepts.

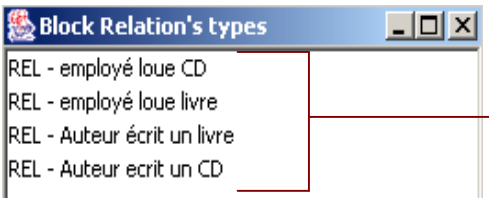


Désigne le début d'une requête ou d'une sous requête

Liste les opérations possibles pour filtrer la requête

Liste les champs de la base de données cible

Liste des relations.



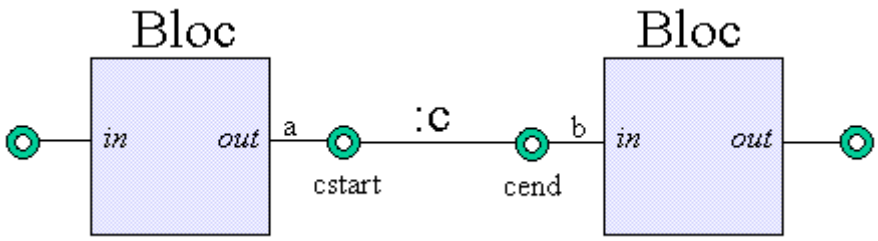
Liste des relations possibles

Ces deux listes sont affichées au démarrage de l'éditeur et créées sur base des données du repository.

Réduction de la grammaire

L'objectif principal de la réduction est de pouvoir plus facilement lire le diagramme pour ensuite construire l'arbre syntaxique.

Pour cela voyons comment simplifier la grammaire de départ en analysant une connexion. Une connexion se construit comme suit :

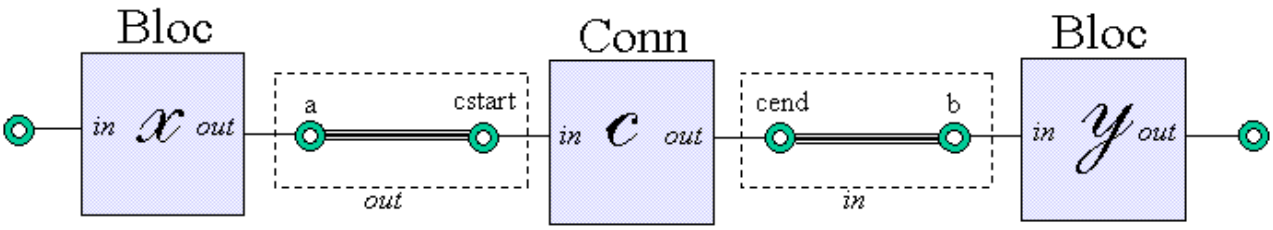


Une connexion 'c' ayant comme extrémités 'cstart' et 'cend' relie deux blocs par leurs connecteurs 'out' et 'in' respectivement appelés 'a' et 'b'.

Si nous regardons du point de vue des connecteurs ‘out’ et ‘in’, nous pouvons déduire qu’il relie chacun des blocs à la connexion ‘c’ de cette manière :

- out (a,cstart)
- in (b,cend)

En allant encore plus loin dans les détails et en remplaçant la connexion ‘c’ par un élément graphique, nous retrouvons de la même manière que pour les blocs, la notion de connecteur. Voici le schéma explicatif :



Dans ce schéma nous avons toujours les relations out(a,cstart) et in(b,cend). Appliquons aussi cette règle sur les connecteurs des deux blocs :

- o : out(x,a)
- i : in(y,b)

Définition de la réduction de la connexion:

Si on considère deux blocs ‘x’ et ‘y’ et une connexion ‘c’, on peut définir la connexion des blocs ‘x’ et ‘y’ appelé *to(x,y)* comme étant égale à l’association de o, i et c.

to(x,y) = o and i and c

Démonstration:

On a au départ: (par associativité)

to(x,y) = out(x,a) and *out(a,cstart)* and *c* and *in(b,cend)* and *in(y,b)*

Si on considère que *c* détient les informations sur *cstart* et *cend*, alors les relations *out(a,cstart)* et *in(b,cend)* peuvent disparaître.

On a alors :

to(x,y) = out(x,a) and *c* and *in(y,b)*

puis

to(x,y) = o and i and c avec *o = out(x,a) et i = in(y,b)*

Traduction de cette règle dans le fichier de grammaire :

‘to’ sera interprété par une classe java qui détiendra les instances des connecteurs ‘o’ et ‘i’. Comme dit ci-dessus ‘c’ sera aussi une classe java et contiendra les instances de ses connecteurs ‘cstart’ et ‘cend’.

```
reducer {
  b:block(x)
  ==>
  f:fb(x) {
    f.block = b.self();
```

```
  };

  c:conn(cstart,cend)
  out(a,cstart) in(b,cend)
  o:output(x,a) i:input(y,b)
  bl1:block(x) bl2:block(y)
  -{ in(b,_) }
  ==>
  t:to(x,y) {
    t.fromConn = bl1.getConnector(o);
    t.toConn = bl2.getConnector(i);
    t.conn = c.self();
  }
  if Semantics.matchingTypes(o,i);

}
```

Contraintes liées au diagramme

Les seules contraintes ajoutées sont la vérification dans la partie ‘reducer’ de la sémantique. Ceci se situe dans le fichier par cette ligne :

if Semantics.matchingTypes(o,i);

Ce qui signifie que la réduction se fera que si les connecteurs sont du même type. Ceci, par exemple, aura pour avantage de ne pas pouvoir lier une relation au connecteur ‘*start query*’ d’un concept.

Annexe C : Implémentation de l'arbre syntaxique

L'arbre syntaxique est construit au moment où le composant d'analyse de diagramme de DIAGEN appelle la méthode 'addConn' de la classe 'Semantics'. Ceci peut être compris dans la grammaire que nous avons réalisée dans l'annexe B. Le composant sait qu'il doit l'appeler car cette action est indiquée dans le fichier de grammaire de Diagen. Ce qui signifie que l'arbre est construit à chaque fois que la grammaire est vérifiée ce qui veut dire à chaque manipulation sur l'éditeur. Toutefois pour construire l'arbre il faut que la grammaire soit correcte.

L'arbre contiendra des connexions. Comme indiqué dans le 'reducer', une connexion est créée sur base du connecteur précédent et suivant et de la connexion proprement dite. Ce qui donnera pour le premier exemple simple ci-dessus 'Liste des compositeurs qui ont créé un CDROM' :

Voici la trace laissée par l'éditeur lors de l'affichage de la requête :

```
[
connector - REQUETE.start query-->COMPOSITEUR.start query,
COMPOSITEUR.Rel. Int. -out-->REL - COMPOSITEUR CREE.COMPOSITEUR,
REL - COMPOSITEUR CREE.CDROM-->CDROM.Rel. Int. –in
]
```

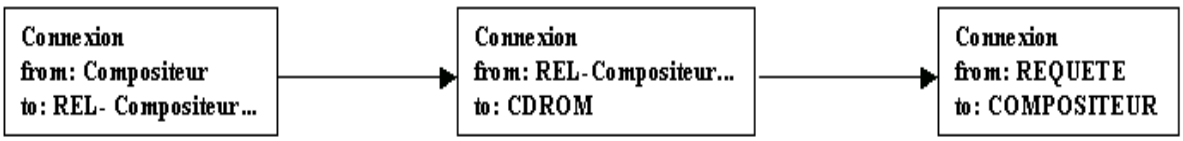
L'arbre est mémorisé sous forme de vecteur. '[' précise le début du vecteur et ']' la fin du vecteur. Chaque connexion est séparée par une virgule. Une connexion est représentée dans la trace par :

'description du bloc *from*' . description du connecteur

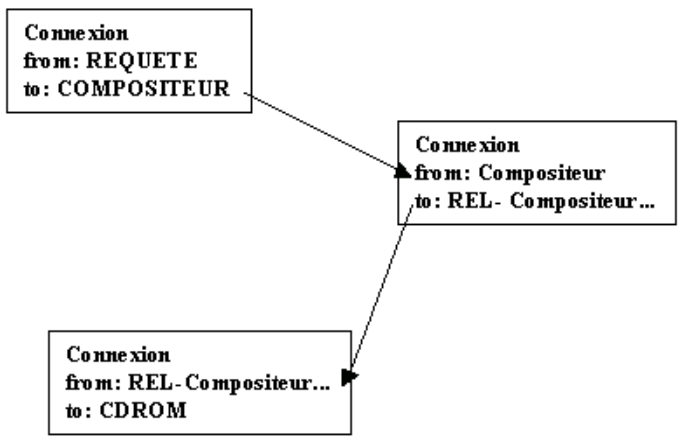
-->

'description du bloc *to*' . description du connecteur

Schéma du vecteur pour la trace ci-dessus :



Le vecteur de DIAGEN suit la représentation graphique de l'écran. De ce fait pour construire l'arbre il faut trouver la connexion ayant le bloc *from* égale à REQUETE. A partir de là nous pouvons construire l'arbre, ce qui donne :



Plus compliqué est l'exemple de jointure intitulé ci-dessus 'Liste des employés compositeurs de CDROM et qui ont loué des CDROM'

Voici la trace obtenue en composant la requête :

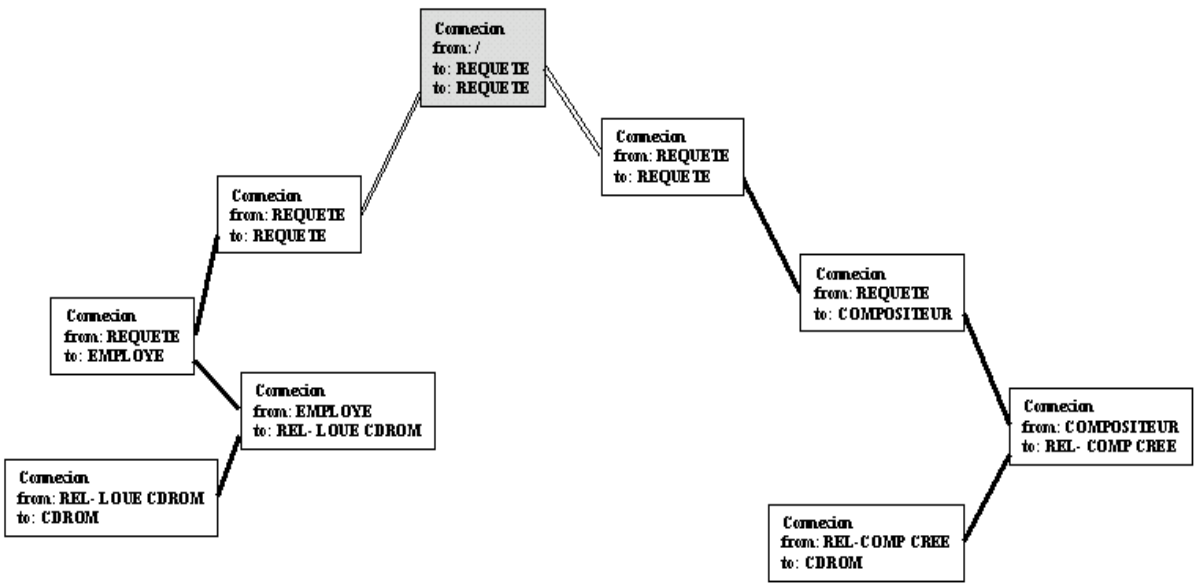
```
[
EMPLOYE.Rel. Int. -out-->REL - LOUE CDROM.EMPLOYE,
REL - LOUE CDROM.CDROM-->CDROM.Rel. Int. -in,
connector - REQUETE.start query-->COMPOSITEUR.start query,
connector - REQUETE.start query-->connector - REQUETE.in,
REL - COMPOSITEUR CREE.CDROM-->CDROM.Rel. Int. -in,
connector - REQUETE.start query-->connector - REQUETE.in,
COMPOSITEUR.Rel. Int. -out-->REL - COMPOSITEUR CREE.COMPOSITEUR,
COMPOSITEUR.Rel. Ext. -out-->operator - =.in,
connector - REQUETE.start query-->EMPLOYE.start query,
operator - =.out-->EMPLOYE.Rel. Ext. –in
]
```

Les traces soulignées indiquent les relations externes de la question formulée par l'utilisateur. Ceci veut dire que deux arbres seront gérés : un arbre représentant les relations internes et un arbre représentant les relations externes.

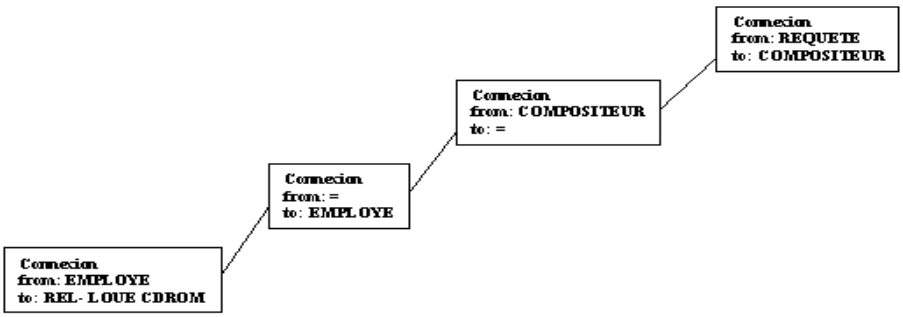
Dans la réalité, la grammaire implémentée dans DIAGEN crée un seul arbre puisque celui des relations externes a toujours comme connexion de début et de fin, une connexion de l'arbre des relations interne. Seulement un arbre ne peut contenir de cycle (boucle) donc il préférable pour la compréhension de les voir en deux arbres distincts. Le composant convertisseur de l'architecture conceptuelle devra faire la différence entre une relation interne et externe. Le moyen mis à sa disposition est le type de relation comme indiquée dans la trace : « Rel. Int » pour une relation interne et « Rel. Ext. » pour une relation externe.

Voici les arbres obtenus à partir de cette trace :

Arbre des relations internes :



Arbre des relations externes :



La connexion représentée sur fond gris de l'arbre des relations internes correspond à une classe qui devra être ajoutée pour obtenir une racine à cet arbre.

La connexion conserve plus que les connecteurs *from* et *to*. Voici la définition de cette classe :

'Conn' est la classe contenue dans le vecteur. Cette classe Java est notre connexion dans les lignes ci-dessus.

```
public class Conn {
	public final Connection conn;
	public final Block fromBlock, toBlock;
	public final Connector fromConn, toConn;

	Conn(
		Connection conn,
		Block fromBlock,
		Connector fromConn,
		Block toBlock,
		Connector toConn) {
		this.conn = conn;
		this.fromBlock = fromBlock;
		this.fromConn = fromConn;
		this.toBlock = toBlock;
		this.toConn = toConn;
		System.out.println(this);
	}

	public String toString() {
		//++++++++++++++++++++++++
		//Affichage des connexions
		//++++++++++++++++++++++++
		return fromBlock.toString() + "." + fromConn.name +
		"-->" +
		toBlock.toString() + "." + toConn.name;
		}
}
```

Cette classe contient trois autres types de classes : Block, Connector et Connection. Les classes Block et Connection correspondent aux classes définies dans le fichier de grammaire et construite par le framework. Nous avons dû surcharger ces classes pour qu'elles affichent la représentation visuelle d'un bloc

(un rectangle avec un titre et une zone propriété) et d'une connexion (un trait) Pour le connector c'est différent, il est là par défaut dans DIAGEN puisqu'un composant (bloc et connexion) possède au moins une zone d'attachement. Un connector correspond à une zone d'attachement donc la définition de la classe et très simple :

```
public class Connector implements java.io.Serializable {
    public final String name;
    public final Type type;

    public Connector ( String name ){
        this(name,Type.NULL);
    }

    public Connector ( String name, Type type ) {
        this.name = name;
        this.type = type;
    }

    public String toString() {
        return name + "[" + type +       "]";
    }
}
```

La sémantique est basée sur l'égalité entre les types des connectors.

Annexe D : Fichier de grammaire inséré dans DIAGEN

Contenu du fichier ‘spec’ inséré dans le répertoire implementation/src/composant:

```
// -*- spec -*-
package composant;

component block[*] { Block{input, output} },
        conn[2] { Connection[Start,End] };

relation in[2]  { Relations.intersect(input,End) },
       out[2] { Relations.intersect(output,Start) };

terminal fb[1] { Block block; },
       to[2] { Connector fromConn, toConn; Connection conn; },
       loop[1] { Connector fromConn, toConn; Connection conn; };

nonterminal Chart[0] { Chart chart; },
          Block[1] { Block block; Chart chart; };

constraintmanager diagen.editor.param.CustomizedConstraintMgr;

reducer {
   b:block(x)
   ==>
   f:fb(x) {
      f.block = b.self();
   };

   c:conn(cstart,cend)
   out(a,cstart) in(b,cend)
   o:output(x,a) i:input(y,b)
   bl1:block(x) bl2:block(y)
   -{ in(b,_) }
   ==>
   t:to(x,y) {
      t.fromConn = bl1.getConnector(o);
      t.toConn = bl2.getConnector(i);
      t.conn = c.self();
   }
   if Semantics.matchingTypes(o,i);

   c:conn(cstart,cend)
   out(a,cstart) in(b,cend)
   o:output(x,a) i:input(x,b)
   bl:block(x)
   -{ in(b,_) }
   ==>
   t:loop(x) {
      t.fromConn = bl.getConnector(o);
      t.toConn = bl.getConnector(i);
      t.conn = c.self();
   }
   if Semantics.matchingTypes(o,i);
}

grammar {
   start Chart<chart>;
```

```
   ch:Chart() ::= << b:Block(_) >> {
                     ch.chart =
Semantics.createChart(<<b.block>>);
                  b.chart = ch.chart;
                   };

   bl:Block(a) ::= b:fb(a) {
                  bl.block = b.block;
              };

   embed t:to(a,b) into bl1:Block(a) * bl2:Block(b) {
                       Semantics.addConn(bl1.chart,
                                         t.conn,
                                    bl1.block, t.fromConn,
                                    bl2.block, t.toConn);
                        };

   embed l:loop(a) into bl:Block(a) * {
                       Semantics.addConn(bl.chart,
                                         l.conn,
                                    bl.block, l.fromConn,
                                    bl.block, l.toConn);
                        };
}
```