HAL
open science

# le support de domaines complexes dans SABRINA: une approche par integration d'un interpreteur LISP

G. Kiernan, R. Le Maoult, F. Pasquer

▶ To cite this version:

G. Kiernan, R. Le Maoult, F. Pasquer. le support de domaines complexes dans SABRINA: une approche par integration d'un interpreteur LISP. RR-0671, INRIA. 1987. inria-00075882

HAL Id: inria-00075882
https://inria.hal.science/inria-00075882

Submitted on 24 May 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Rapports de Recherche

N° 671

# LE SUPPORT DE DOMAINES COMPLEXES DANS SABRINA : UNE APPROCHE PAR INTÉGRATION D'UN INTERPRÉTEUR LISP

Gérald KIERNAN
Régine LE MAOULT
Fabrice PASQUER

Mai 1987

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.: (1) 39 63 55 11

# LE SUPPORT DE DOMAINES COMPLEXES DANS SABRINA : UNE APPROCHE PAR INTEGRATION D'UN INTERPRETEUR LISP

Gérald KIERNAN (*), Régine LE MAOULT (**), Fabrice PASQUER (***)

(*) Université PARIS VI
75005 PARIS

(**) Institut d'Informatique d'Entreprise
18, allée Jean Rostand
91000 EVRY

(***) INRIA Rocquencourt
Projet SABRE
BP 105 78153 LE CHESNAY Cedex

## Résumé

Afin d'utiliser le SGBD SABRINA dans le cadre d'applications non classiques (CAO, messagerie, cartographie, textuelles...), une extension du système pour supporter des domaines complexes est proposée. L'extension est basée sur l'adjonction au coeur du système d'un interpréteur LISP. L'administrateur peut alors définir ses propres domaines sous formes de structures et de fonctions LISP associées. Un mécanisme d'héritage de propriétés (fonctions) entre domaines est aussi prévu. Toute valeur d'un domaine complexe est stockée comme une arborescence LISP. Les fonctions utilisateur sont stockées dans une relation particulière. Après définition de domaines complexes, l'utilisateur peut les référencer lors de la définition de ses relations puis les invoquer au travers de requêtes du langage externe.

## Mots clés

domaine complexe, fonction, domaine complexe, relationnel, multi-médias, lisp, héritage, architecture.

*Ces travaux ont été en partie financés par la convention No. 857B077de recherche avec le CNET LANNION.*

# THE SUPPORT OF COMPLEX DOMAINS IN A RELATIONAL DBMS USING A LISP LANGUAGE PROCESSOR

Gerald KIERNAN (*), Régine LE MAOULT (**), Fabrice PASQUER (***)

(*) University of Paris VI
MASI Laboratory
4 Pl Jussieu
75005 Paris

(**) Institut d'Informatique d'Entreprise
18 allée Jean Rostand
91000 EVRY

(***) INRIA Rocquencourt
SABRE Project
BP 105 78153 LE CHESNAY Cedex

## ABSTRACT

This article presents an extension of a DBMS to include complex domains. This extension is currently being implemented in the SABRINA DBMS, a DBMS developed in relation to the SABRE project at INRIA and at the MASI laboratory of Paris VI University. Its main aspect is a LISP interpreter designed as an integrated DBMS processor. We adopt the current notion of Abstract Data Type (ADT) which is: a domain data structure (DDS) with its associated operators and we define the new concept of complex domain as an ADT on which can be applied operator inheritance. All complex domains are represented as LISP structures. New user-defined functions are stored in a meta data base relation. The user can reference new complex domains when creating new relations. He can then define new functions for these domains and use them in the external user language which is an extended version of SQL.

**KEY WORDS :** Complex Domain, Abstract Data Type, Relational Database, LISP.

PAPIER RÉCUPÉRÉ ET RECYCLÉ

# INTRODUCTION

Aujourd'hui, beaucoup d'applications souhaitent utiliser la puissance et la souplesse du modèle relationnel pour gérer des informations généralisées. La simplicité du modèle s'explique par les trois notions de base (domaine, relation, attribut) construites à partir de la théorie des ensembles. Des langages de manipulations de données (LMD) sont définis sur ces structures. La base de ces langages est le calcul des prédicats du premier ordre augmenté des fonctions agrégats et des fonctions arithmétiques. L'implantation actuelle des notions de domaines et de fonctions est insuffisante pour des applications nécessitant la gestion de blocs d'informations importants (graphiques, représentation cellulaire ou polygonale des images,...) ou très structurés (CAO, textes, cartes géographiques,...). En effet les domaines permis sont généralement les domaines de base (entier, réel, chaîne de caractères, booléen) et les opérateurs du langage sont restreints aux opérations classiques sur ces domaines. Il est donc nécessaire d'une part d'implanter plus complètement la notion de domaine en introduisant des ensembles d'objets structurés comme domaines, et d'autre part de généraliser les opérations en permettant la définition de fonctions utilisateurs sur ces domaines.

Des propositions ont déjà été faites pour enrichir les domaines et les opérations dans un contexte relationnel ou dans un modèle différent. Les applications bureautiques ou de gestion de documents sont très représentatives des nouveaux besoins. Le système TIGRE propose des solutions pour ce type d'application [ADIB 85]. L'aspect hiérarchique des données et leur modélisation sous forme de V-relations peut aussi permettre des représentations et des manipulations de données complexes [VERS 86]. Dans [VALD86] une approche semblable a été développée. Dans un contexte plus purement relationnel, les approches des systèmes INGRES [STON 83, 85], et RAD [OSBO 86] sont relativement similaires et visent à définir et implanter la notion de type abstrait et d'opérations sur ces nouveaux types. Les difficultés soulevées par ces solutions sont essentiellement l'intégration des fonctions utilisateurs dans le code du SGBD, et la modification du noyau interne du système. Une option consistant à implanter un module au dessus du SGBD classique et réalisant des modifications de schémas et de questions[TSUR 84], [ZANI 83, 85] ne modifie pas le SGBD, mais pose des problèmes de performances liées aux nombreuses opérations de construction nécessaires.

Notre but est de conserver le modèle relationnel et d'ouvrir le système à la définition de domaines complexes et à leurs manipulations. L'approche originale proposée dans SABRINA est basée sur l'intégration d'un interpréteur LISP au SGBD. Cet interpréteur permet de définir en LISP les nouveaux domaines et fonctions associées. La spécification en LISP de la composition du domaine permet d'assurer la vérification des occurences des valeurs d'attribut. Les contraintes d'intégrité de domaines complexes peuvent être ainsi exprimées. En ce qui concerne les fonctions,

la spécification des hiérarchies de spécialisations va permettre de rattacher un ensemble d'opérateurs à un domaine donné et d'appliquer l'héritage de ses opérateurs à des sous domaines. Finalement les fonctions utilisateurs étant interprétées, les erreurs peuvent être facilement contrôlées. L'approche est donc originale puisqu'elle permet l'extensibilité du SGBD par du code écrit par l'usager et contrôlé lors de l'exécution des requêtes, la vérification systématique de l'appartenance des objets aux domaines complexes (contraintes d'intégrité de domaines), le support de fonctions quelconques au niveau du langage d'interrogation. Par exemple, sur un domaine complexe POLYGONE comportant les fonctions :

- SURFACE permettant de calculer la surface,
- TRACER permettant l'édition graphique,

l'utilisateur pourra définir une relation représentant le cadastre comme suit :

CREATE CADASTRE (NO_PARCELLE : ENTIER, REPRESENTATION : **POLYGONE**) ;

Il pourra ensuite invoquer les fonctions au travers du langage externe étendu, afin par exemple de tracer les parcelles de surface supérieure à 5000 $m^2$, comme suit :

```
SELECT  NO_PARCELLE, REPRESENTATION.TRACER
 FROM   CADASTRE
WHERE  REPRESENTATION.SURFACE > 5000 ;
```

Cet article présente donc l'intégration des domaines complexes dans le SGBD SABRINA. Cette intégration est possible grâce aux fonctionnalités du langage LISP, et à la possibilité d'étendre le calcul de tuples de SABRINA avec des fonctions écrites en LISP. La première section présente la notion de domaine complexe. La section suivante montre comment l'intégration de l'interpréteur LISP permet la définition des domaines et des opérations. Les propriétés d'héritage liées aux hiérarchies de domaines et l'aspect intégrité sont aussi développés. Des exemples de manipulation de données complexes au sein de requêtes du langage externe sont ensuite présentées. Finalement l'implantation actuelle et future de l'interpréteur LISP dans le système SABRINA est montrée ainsi que la manière dont sont traitées les requêtes externes comportant des fonctions complexes. La conclusion donne les points clés sur lesquels nous travaillons aujourd'hui : intégrité, fiabilité, performance.

## 1. DOMAINE COMPLEXE

Les concepts de base du modèle relationnel, et en particulier ceux de domaine et de relation, n'impliquent aucune restriction sur le type des occurences de valeurs d'un domaine. Un domaine est un ensemble de valeurs, une relation est un sous-ensemble du produit cartésien d'une liste de

domaines. Un domaine peut très bien être défini comme l'ensemble des entiers ou comme l'ensemble des employés d'une entreprise. Dans les systèmes commercialisés, l'implantation des domaines simples (entier, réel, chaîne de caractères, booléen) a permis de montrer la faisabilité des concepts généraux du modèle relationnel. La gestion, le stockage et la manipulation des domaines de base sont aujourd'hui bien définis. Les applications du type gestion traitent des informations simples et généralement de longueur fixe. Cependant, actuellement, l'utilisation des SGBD se diversifie et la notion de domaine doit être reconsidérée.

L'idée première est d'enrichir les domaines supportés pour prendre en compte les nouvelles applications utilisant les SGBD comme la CAO, la bureautique, la cartographie. Les structures des données de ces applications sont plus complexes et variées : structure hiérarchique pour les applications CAO, représentation géométrique en bases de données géographiques,... La définition de domaines complexes doit donc permettre en particulier la spécification de domaines hiérarchiques, d'agrégations de domaines,... Une définition de domaine peut alors être élaborée à partir de produits cartésiens de domaines déjà définis, ceci afin de supporter des domaines multiformes. Il reste encore à spécifier les domaines prédéfinis qui serviront de constructeurs de domaines (liste, ensemble, graphe...) , tel qu'il est établi dans SMALLTALK [GOLD 83]. Ainsi, on obtient le langage de définition approché suivant :

<domaine de base> ::= entier | réel | chaine de caractères | booléen ;
<domaine> ::= { <domaine de base> | <domaine multiforme> }
et <caractéristiques> ;
<domaine multiforme> ::= <domaine> | <domaine> X <domaine multiforme> ;

< caractéristiques > ::= ensemble de critères ou fonctions que doivent vérifier
les objets pour appartenir au domaine.

Exemples de domaines :

D-POINT = ENTIER X ENTIER ;
D-POLYGONE = D-POINT | D-POINT X D-POLYGONE ;
D-TRIANGLE = D-POINT X D-POINT X D-POINT ;
D-TRIANGLE_RECTANGLE = D-TRIANGLE
et < il existe un angle droit > ;

La clause < il existe un angle droit > est une caractéristique du domaine D-TRIANGLE_RECTANGLE. Un domaine peut être défini comme une spécialisation d'un autre domaine en ajoutant simplement des caractéristiques ou redéfini complétement. Par exemple, le domaine D-TRIANGLE_RECTANGLE aurait pu être défini directement comme suit :

Domaine D-TRIANGLE_RECTANGLE = D-POINT X D-POINT X D-POINT
et < il existe un angle droit > ;

Jusqu'à présent, seule la notion de domaine a été spécifiée. La définition d' un type d'objet peut maintenant être précisée. Un type d'objet est "un ensemble d'objets possédant des caractéristiques similaires et manipulables par des opérations identiques" [Gard 83]. Un type d'objet est défini par un domaine et par les opérations (ou fonctions) de manipulation associées à cet objet. Les exemples suivants montrent des types utilisant les domaines précédemment définis.

Type POINT :
Domaine = ENTIER x ENTIER ;
Fonctions = Abscisse,
Ordonnée,
Distance_à_origine, ...

Type POLYGONE :
Domaine = POINT | POINT x POLYGONE ;
Fonctions = Périmètre,
InclusdansSurface,
Tracer, ...

Type TRIANGLE :
Domaine = POINT x POINT x POINT
Fonctions = Surface
Isocèle,
Equilatéral, ...

Type TRIANGLE_RECTANGLE :
Domaine = TRIANGLE
et < il existe un angle droit > ;
Fonctions = Hypothnuse
Surface
Isocèle, ...

Les fonctions associées aux types d'objets sont définies par l'utilisateur. La puissance de la notion de type est très importante puisqu'elle permet l'héritage de propriétés. Prenons l'exemple des

domaines TRIANGLE et TRIANGLE_RECTANGLE définis par un usager, il serait utile que cet utilisateur n'ait pas à redéfinir au niveau TRIANGLE_RECTANGLE les fonctions qu'il a spécifiées au niveau TRIANGLE comme Surface, Isocèle. On adopte donc la notion de généralisation et spécialisation [NILS 81] : tout type qui est une spécialisation d'un autre type hérite des fonctions de ce type. Ainsi, si l'usager spécifie que le type TRIANGLE_RECTANGLE est une spécialisation du type TRIANGLE, alors il pourra appliquer les fonctions Surface et Isocèle définies dans TRIANGLE pour un TRIANGLE_RECTANGLE.

On peut se demander si cette définition de type d'objet est conforme avec celle donnée dans [GUTT 77] : "un type abstrait de donnée est une définition formelle de chaque opération applicable à un type de donnée, sans tenir compte de la représentation du type". Dans notre modèle, l'ensemble des opérateurs associés à un type peut être en effet précisé sans connaître la représentation interne du type. Donc, un utilisateur n'a pas besoin de se soucier de la manière dont sont implantés les domaines et fonctions pour les utiliser par le langage assertionnel ou dans un programme d'application.

Dans le paragraphe suivant le système SABRINA-LISP est présenté et en particulier le langage SIMPLEX (Sabre Intégrant des Manipulations d'objets comPLEX) qui permet la définition des types d'objets et leur utilisation au sein des requêtes.

## 2. LE LANGAGE DE DEFINITION ET DE MANIPULATION D'OBJETS COMPLEXES

Le paragraphe précédent a défini ce que sont les types de données complexes dans SABRINA. En fait, les objets d'un certain domaine ne sont manipulables que par des fonctions. Il est alors nécessaire de disposer d'un langage de définition de données qui intègre des aspects fonctionnels, le langage LISP correspond tout à fait à ce besoin. L'approche fonctionnelle répond aux objectifs visés dans l'introduction des domaines complexes, à savoir une approche générale et thésaurisante pouvant répondre à tout genre de problème de représentation et de traitement. L'interpréteur LISP intégré à SABRINA présente un point d'entrée par lequel de nouveaux domaines et de nouvelles fonctions sont définis et enregistrés dans le SGBD. En plus d'offrir un environnement de programmation, le langage LISP offre la structure de liste pour construire les objets structurés nécessaires à la manipulation de domaines complexes.

Le langage LISP est à l'origine un langage non typé. C'est seulement au moment de l'évaluation de la fonction que la cohérence de l'argument vis à vis de la fonction est vérifiée. Une fonction qui évalue une liste de valeurs numériques ne serait interrompue que s'il se présentait dans

la liste un symbole ou une autre liste. Autrement, l'évaluation se ferait sans problème qu'il s'agisse d'une liste de salaires ou de coordonnées d'un objet géométrique. Pour être compatible avec les exigences de cohérence d'un SGBD, nous avons introduit la notion de types dans l'interpréteur LISP.

## 2.1. La définition de types

L'introduction de nouveaux types permet à la fois de spécifier la composition du domaine et des fonctions associées à ce domaine. La spécification des généralisations permet de rattacher un ensemble de fonctions à un domaine et d'appliquer l'héritage de ses fonctions aux sous-domaines. La spécification d'un domaine comporte : le nom du domaine générique appelé simplement généralisation, son nom spécifique et sa définition. La généralisation permet d'appliquer le principe d'héritage de fonctions. Le nom spécifique permet de lui associer des opérateurs et sa définition assure l'intégrité des valeurs du domaine. A chaque définition de domaine correspond une fonction LISP particulière. Ces fonctions seront des fonctions de type *dd* (definir domaine) et vérifie qu'un objet appartient à un domaine. La structure générale de la fonction *dd* est la suivante :

*(dd <nom domaine générique> <nom domaine> (<objet à vérifier>)*
*(<définition du domaine>))*

LISP est le nom du domaine de plus haut niveau de la hiérarchie de généralisation. Voir figure 2.1. Ce domaine comporte un ensemble d'opérateurs de base applicables à tous les domaines complexes : par exemple, les fonctions CAR et CDR. Le domaine LISP est ainsi prédéfini :

```
(dd NIL LISP (X)
 (or (atom x) (listp x)))
```

Selon cette définition, X est soit un atome soit une liste, cette définition sera vraie pour tout objet représenté par le langage LISP.

Si le domaine NIL est déclaré dans la partie généralisation d'un domaine, alors celui-ci n'hérite d'aucun opérateurs d'un autre domaine.

La fonction LISP permettant de définir un point cartésien est :

```
(dd LISP POINT (X)
 (and (listp x)
      (numberp (car x))
      (numberp (car (cdr x)))
      (null (cdr (cdr x)))))
```

Selon cette définition, pour qu'un objet X soit un point, il doit être une liste dont le premier et le deuxième élément sont des nombres. Le domaine point est sous-domaine du domaine LISP. Dans cet exemple, tous les opérateurs du domaine LISP sont applicables au domaine point.

La définition d'un polygone est construite récursivement à partir de celle d'un point :

```
(dd LISP POLYGONE (X)
 (cond
  ((null x) nil)
  ((and (point (car x)) (null (cdr x))) t)
  (t (and (point (car x)) (polygone (cdr x))))))
```

Même si le domaine polygone est construit à partir de la définition d'un point ce n'est pas une spécialisation d'un point : il ne faut pas confondre la hiérarchie de généralisation et la possibilité de définir une fonction LISP à partir d'autres fonctions LISP existantes. Le domaine polygone est déclaré comme une spécialisation du domaine LISP afin de pouvoir appliquer sur les polygones les opérateurs de base inclus dans le domaine global LISP.

Toute fonction applicable à un polygone est applicable à un quadrilatère. Un quadrilatère est une spécialisation d'un polygone et va donc hériter de ses opérateurs. La définition du domaine du quadrilatère est la suivante :

```
(dd POLYGONE QUADRILATERE (X)
 (and
  (point (car x))
  (point (car (cdr x)))
  (point (car (cdr (cdr x))))
  (point (car (cdr (cdr (cdr x)))))
  (null (cdr (cdr (cdr (cdr x)))))))
```

Un triangle est une spécialisation d'un polygone et est composé de trois points. Un triangle peut être définit comme suit :

```
(dd POLYGONE TRIANGLE (X)
 (and (point (car x))
  (point (car (cdr x)))
  (point (car (cdr (cdr x))))
  (null (cdr (cdr (cdr x))))))
```

Un triangle isocèle est un triangle qui comporte deux côtés égaux. Un triangle isocèle est une spécialisation d'un triangle avec une caractéristique : la fonction isocèle définie sur un triangle.

```
(dd TRIANGLE TRI-ISOC (X)
(and (triangle x) (isocele x)))
```

Un opérateur (ou fonction) est défini à l'aide de la fonction *de* ou la fonction *df*. Contrairement à la fonction *dd*, les fonctions *de* et *df* sont des fonctions standard dans les versions classiques de LISP. La structure générale de ces deux fonctions est la suivante :

```
(de  <domaine de résultat> <nom fonction> (<nom domaine> (<liste de variable>)
            [<nom domaine> (<liste de variable>)]* )
    (<définition de la fonction>))
```

*Nom fonction* est le nom symbolique de la fonction qui porte sur des objets de domaine *nom domaine*, et qui retourne une valeur de domaine "*domaine de résultat.*". Plusieurs objets et plusieurs paramètres peuvent être référencés dans une fonction.

Les fonctions suivantes permettent de déterminer si un triangle est un triangle isocèle. La fonction ISOCELE est une fonction qui porte sur les triangles :

```
(de BOOLEEN ISOCELE (TRIANGLE (X))
 (calcisoc
  (car (car x)) (car (cdr (car x)))
 (car (car (cdr x))) (car (cdr (car (cdr x))))
 (car (car (cdr (cdr x)))) (car (cdr (car (cdr (cdr x)))))))

(de  BOOLEEN CALCISOC ( ENTIER (A1 A2 B1 B2 C1 C2))
(or
 (equal (XY2 A1 A2 B1 B2) (XY2 A1 A2 C1 C2))
 (equal (XY2 A1 A2 B1 B2) (XY2 B1 B2 C1 C2))
 (equal (XY2 A1 A2 C1 C2) (XY2 B1 B2 C1 C2))))
(de ENTIER XY2 (ENTIER (X1 X2 Y1 Y2))
 (plus (sqr (moins X1 Y1)) (sqr (moins X2 Y2))))
```

Les fonctions CALCISOC et XY2 servent uniquement au calcul de la fonction ISOCELE.

Les fonctions suivantes permettent de calculer la surface d'un triangle quelconque :

```
(de ENTIER SURFACE (TRIANGLE (X))
(div (sqrt
(calcsurf
   (car (car x))(car (cdr (car x)))
   (car (car (cdr x)))(car (cdr (car (cdr x))))
   (car (car (cdr (cdr x))))(car (cdr (car (cdr (cdr x)))))))) 2))

(de ENTIER CALCSURF (ENTIER (a1 a2 b1 b2 c1 c2))
(plus
  (mul (sqr a1)(sqr b2))(mul (sqr a1)(sqr c2))(mul (sqr a2)(sqr c1))
  (mul (sqr b1)(sqr c2))(mul (sqr b2)(sqr c1))(mul (sqr a2)(sqr b1))
  (mul -2(sqr a1)b2 c2)(mul -2(sqr a2)b1 c1)(mul -2(sqr b1)a2 c2)
  (mul -2(sqr b2)a1 c1)(mul -2(sqr c1)a2 b2)(mul -2(sqr c2)a1 b1)
  (mul 2 a2 b1 b2 c1)(mul 2 a1 b1 b2 c2)(mul 2 a2 b1 c1 c2)
  (mul 2 a1 b2 c1 c2)(mul 2 a1 a2 b2 c1)(mul 2 a1 a2 b1 c2)
  (mul -2 a1 a2 b1 b2)(mul -2 a1 a2 c1 c2)(mul -2 b1 b2 c1 c2)))
```

La fonction SURFACE calcule la surface pour des objets de domaine triangle.

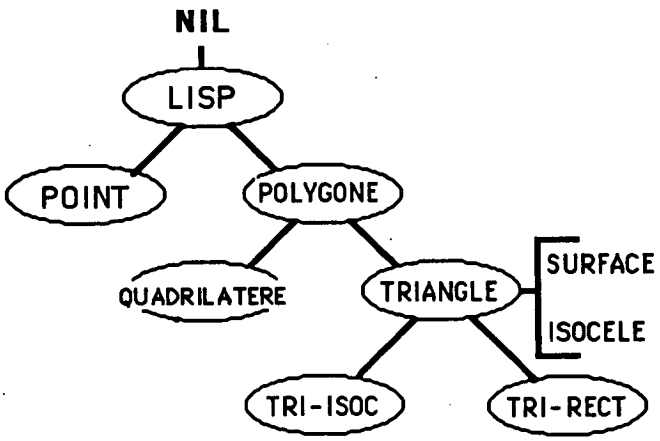


**Figure 2.1. Hiérarchie des domaines donnés en exemples**

La figure 2.1 est une représentation graphique de la hiérarchie des domaines. Lors de la définition d'un nouveau domaine, celui-ci s'insère à la bonne place dans cette hiérarchie. Un noeud hérite des fonctions de ses parents mais pas de ses frères. Par exemple, un triangle isocèle et un triangle rectangle héritent des fonctions surface et isocèle. Mais le domaine triangle-isocèle n'hérite par des

fonctions définies sur triangle-rectangle et inversement.

## 2.2. Le langage de manipulation de données

Le langage externe permettant de travailler sur les données complexes est construit à l'aide d'un ATN [PASQ 85], [ELM 85], analysant un langage proche de SQL, auquel sont rajoutées des fonctionnalités liées à l'introduction des domaines complexes. Ce langage inclut dans ses manipulations de base les commandes de manipulation de données (recherche, insertion, suppression, mise à jour), les commandes de définition de données et les commandes de définition de contrôles. La façon de définir les nouveaux domaines (domaines et fonctions associées) a été présentée ci-dessus. Nous nous intéressons ici à l'utilisation au niveau externe de ces nouvelles fonctions et donc à leur intégration au sein des requêtes de manipulation de données. La syntaxe de base des différentes requêtes est exprimée par les clauses SQL étendues quand c'est nécessaire.

Nous illustrons le langage par une relation décrivant des triangles avec trois attributs : le numéro de triangle NT de domaine entier, sa description de domaine texte et ses coordonnées de domaine TRIANGLE, comme défini ci-dessus.

Exemple de tuple de la relation TRIANGLES:

| | |
|---|---|
| NT : | 1 |
| DESCRIPTION : | triangle-rectangle |
| COORDONNEES : | ( ( 1 1 ) ( 8 1 ) ( 8 8 ) ) |

L'insertion de ce tuple dans la relation TRIANGLES peut s'exprimer :

INSERT INTO TRIANGLES ( 1 , triangle-rectangle , ((1 1) (8 1) (8 8)) ) ;

L'expression d'une valeur d'un domaine complexe est ici simple car elle est exprimée de manière très concise et sous la forme d'un texte. Il sera important de pouvoir invoquer des fonctions LISP afin de permettre la saisie de valeurs plus complexes (graphiques, cartes,..). Néanmoins cet exemple est intéressant car la manipulation de ces valeurs reste indépendante de la manière dont sont saisies les informations.

Une fonction définie sur un domaine peut être employée dans toutes les clauses d'une requête (projection, restriction, agrégation ou tri). Pour la manipulation de ces fonctions, le calcul de tuples avec fonctions tel qu'il est défini dans [ULLM 80] est utilisé. La spécification d'un attribut est une fonction appliquée à la variable tuple associée à une relation. Etant donné que les fonctions

utilisateurs définies précédemment s'appliquent sur des résultats de projection d'un tuple, nous utiliserons une notation fonctionnelle permettant l'expression de composition de fonctions (fonctions standard et fonctions utilisateurs). Ainsi, l'expression d'une fonction F sur un attribut A d'une relation X (A étant lui même une fonction appliquée à la variable X) se note X.A.F. Les fonctions utilisateurs se composent les unes avec les autres et acceptent des paramètres en nombre quelconque. L'expression résultante devient $X.A.F_1(p^1{}_1,...,p^1{}_n).F_m(p^m{}_1,...,p^m{}_n)$, où $F_1,...,F_m$ sont des fonctions utilisateurs composées les unes avec les autres (notation pointée) comportant des paramètres $p^1{}_1,...,p^m{}_n$. Il est aussi possible d'appliquer ces fonctions sur un ensemble d'attributs avec la notation suivante : $(X.A_1, Y.A_2, ...,Z.A_n). F_1(...) . F_n (...)$.

Les exemples suivants présentent quelques aspects des nombreuses possibilités offertes.

**Exemple 1**: Utilisation d'une fonction dans la projection.

Rechercher la surface du triangle numéro 3.

```
SELECT   COORDONNEES.SURFACE
 FROM    TRIANGLES
WHERE    NT = 3 ;
```

**Exemple 2** : Utilisation d'une fonction paramétrée dans le résultat.

Visualisation en rouge des triangles rectangles dont la surface est supérieure à 100.

```
SELECT  COORDONNEES.TRACER (ROUGE)
 FROM   TRIANGLES
WHERE   COORDONNEES.SURFACE > 100
        AND COORDONNEES.RECTANGLE = 'Vrai' ;
```

**Exemple 3** : Utilisation globale des fonctions.

Rechercher les triangles de surface inférieure à la surface du triangle numéro 12. On précise aussi la surface de ces triangles dans la projection.

```
SELECT   *, COORDONNEES.SURFACE
 FROM    TRIANGLES
WHERE    COORDONNEES.SURFACE <
         SELECT   COORDONNEES.SURFACE
          FROM    TRIANGLES
         WHERE    NT = 12 ;
```

**Exemple 4** : Utilisation des fonctions dans les agrégats.

Rechercher la somme des surfaces des triangles 1 et 2.

```
SELECT   SOM ( COORDONNEES.SURFACE)
 FROM    TRIANGLES
WHERE    NT  IN ( 1, 2 ) ;
```

En fait l'agrégat tel qu'il est défini dans les langages assertionnels est une fonction standard du système qui est appliquée sur un ou plusieurs attributs d'un ensemble de tuples. Les exemples de fonctions ont jusqu'alors porté sur le domaine d'un attribut . Il est aussi possible de définir des fonctions qui ne travaillent plus sur l'occurence d'un attribut mais sur un ensemble d'occurences de l'attribut tout comme les fonctions agrégats standard. Soit la fonction RASSEMBLE définie sur le domaine TEXTE créant un texte comportant toutes les occurences de valeurs différentes de l'attribut. Appliquée sur l'attribut DESCRIPTION de la relation TRIANGLES, la fonction RASSEMBLE donne pour résultat toutes les descriptions de triangles (triangle-rectangle triangle-isocèle...). Ainsi dans un système comme SABRINA-LISP les agrégats prédéfinis peuvent être absents puisqu'il est possible de les définir par le langage de définition de fonctions. Cette remarque est intéressante car nous verrons dans le paragraphe suivant que l'implantation future met au même niveau le traitement des agrégats et des fonctions LISP et donc permet cette généralisation.

Une autre fonctionnalité, liée à l'héritage de fonctions, est de pouvoir directement utiliser les fonctions LISP de base dans le langage de requêtes. Le domaine TRIANGLE est défini dans l' exemple comme une spécialisation du domaine LISP et donc il hérite des fonctions standard du langage LISP. Il est possible d'appliquer les fonctions CAR, CDR,... sans avoir à redéfinir et renommer ces fonctions. Prenons l'exemple d'une fonction PREMIERPOINT définie sur le domaine TRIANGLE et donnant les informations concernant ce premier point. Elle peut être ainsi créée:

```
(de POINT PREMIERPOINT (TRIANGLE (X))
( CAR  X))
```

La requête de suppression des triangles dont le premier point est le point origine peut soit utiliser la fonction PREMIERPOINT, soit directement utiliser la fonction CAR.

**Exemple 5** : Supprimer les triangles dont le premier point est l'origine.

```
DELETE
FROM   TRIANGLES
WHERE  COORDONNEES.PREMIERPOINT = ( 0 0 ) ;

DELETE
FROM   TRIANGLES
WHERE  COORDONNEES.CAR = ( 0 0 ) ;
```

# 3. IMPLANTATION DU PROCESSEUR DE GESTION DE DOMAINES COMPLEXES

Ce paragraphe présente l'architecture actuelle de SABRINA-LISP et l'exécution de requêtes comportant des fonctions sur des domaines complexes. Une architecture plus optimisée du point de vue performance est ensuite proposée.

## 3.1. Architecture actuelle de SABRINA-LISP

Le système SABRINA est divisé en trois machines : la machine interne, appelée machine algébrique qui exécute les commandes de l'algèbre relationnelle étendue, la machine intermédiaire, appelée machine assertionnelle qui exécute les commandes du calcul relationnel de tuples et la machine externe qui offre diverses interfaces aux utilisateurs. Chaque machine est elle-même divisée en une ou plusieurs couches, chaque couche étant composée de un ou plusieurs processeurs logiques, réalisés sous forme de module logiciel.

Le traitement des domaines et des fonctions associées est aujourd'hui réalisé par le processeur de nom PDC (Processeur de Domaines Complexes). Ce processeur intègre l'interpréteur LISP et se situe dans la machine d'interfaces. Aucune modification n'est apportée aux machines assertionnelle et algébrique. La figure 3.1 présente l'architecture de la version actuelle de SABRINA-LISP.

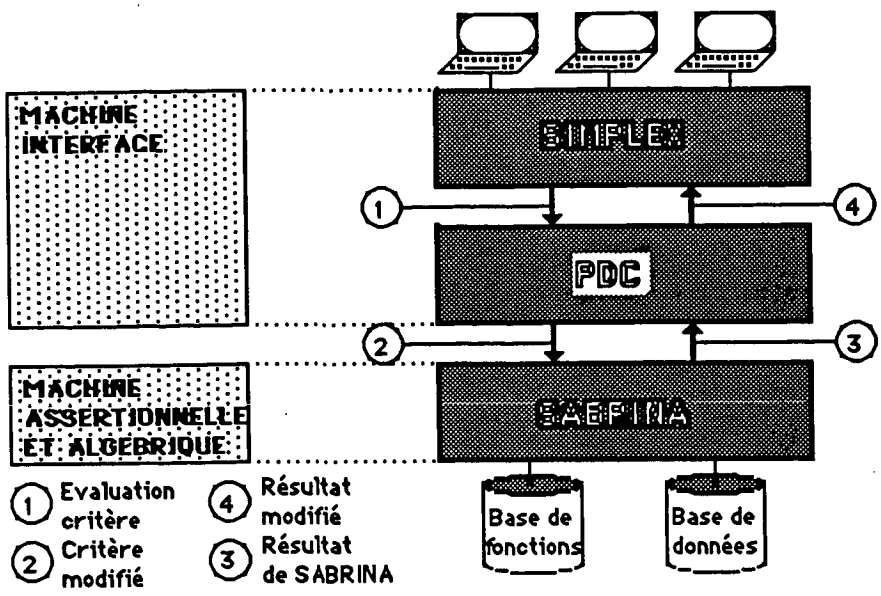


**Figure 3.1. Architecture actuelle de SABRINA-LISP**

Cette architecture a été choisie pour établir, dans un premier temps, la faisabilité d'une telle intégration, sans modifier les couches et les processeurs internes de la machine. Avec ce premier prototype, des requêtes utilisant des fonctions complexes sont traitables. Ces requêtes sont filtrées par le processeur PDC, lequel réalise les fonctions utilisateurs par des techniques de modifications de questions et d'appels à l'interpréteur LISP. Cette solution n'est pas optimale. Dans l'architecture future, l'accès au processeur de gestion des domaines complexes devra être distribué dans les couches et processeurs appropriés de la machine.

## 3.2. L'environnement de programmation

SABRINA-LISP intègre donc un interpréteur LISP conçu pour le traitement et la représentation d'objets complexes. L'interpréteur LISP développé est compatible avec VLISP et LeLISP décrit dans [WERT 85]. Le traitement d'objets symboliques par LISP permet de construire et de sauvegarder de nouveaux objets dont la structure peut être manipulée par l'interpréteur LISP. Son adaptation implique l'implantation d'un ensemble de fonctions non-standard dans les versions courantes du langage. Une *base de fonctions* a donc été créée pour sauvegarder toutes les informations relatives à l'interpréteur. Cette base de fonctions, connue de l'interpréteur, contient les définitions de fonctions et de domaines, des valeurs d'atomes symboliques et des listes de propriétés rattachées à des atomes symboliques.

L'environnement de programmation ainsi totalement intégré au SGBD permet une très grande souplesse dans la définition et l'utilisation des nouveaux outils. En effet les fonctions utilisateurs et

les nouveaux domaines ne sont pas stockés dans des fichiers extérieurs au SGBD mais au contraire dans la base de fonctions gérée par SABRINA. En fait, les diverses relations composant cette base font partie intégrante de la métabase de SABRINA et sont donc accédées et utilisées comme n'importe quelle relation. La principale relation de cette base stocke les définitions de fonctions. Des tuples de cette relation sont présentés figure 3.2. Une fonction est caractérisée par son nom, le nom du domaine sur lequel la fonction est définie, un type de fonction parmi EXPR, FEXPR, CVAL, PLIST, DOMAIN et le code de la fonction.

LISP.FONCTION ( NOM : TEXTE, DOMAINE-ASSOCIE : TEXTE, TYPE -FONCT: TEXTE, DEFINITION : TEXTE )

| | | | |
|---|---|---|---|
| SURFACE | TRIANGLE | EXPR | (CAR(CDR... |
| RECTANGLE | TRIANGLE | EXPR | (QUOTE(CAR... |

**Figure 3.2. Exemples de tuples de la relation LISP.FONCTION**

Avec cette base de fonctions, l'environnement nécessaire reste homogène. L'interpréteur LISP est modifié afin de l'adapter à nos besoins. Nous lui avons ajouté en particulier les fonctionnalités d'accès autonomes à la base de fonctions suivantes :

* Sauvegarde d'une fonction ou d'une définition de domaine;
* Chargement d'une fonction de la base de fonctions;
* Affichage du contenu de la base de fonctions;
* Suppression d'une fonction ou d'un domaine contenu dans la base de fonctions.

Ainsi c'est l'interpréteur qui accède aux relations lors de la définition de fonctions et lors de l'évaluation d'une expression nécessitant le chargement d'une fonction n'étant pas dans l'espace de travail. Les fonctionnalités du langage externe sont aussi étendues afin de permettre la définition de domaines sous forme de fonction booléenne en incluant dans cette définition la référence à un sur-domaine pour appliquer les propriétés d'héritage.

Pour définir une fonction, l'interpréteur LISP est lancé à partir du langage externe du SGBD SABRINA par la commande DEFINIR FONCTION. Pour utiliser des expressions LISP à l'intérieur d'une requête, les fonctions doivent avoir été préalablement définies. La commande d'accès à l'interpréteur LISP se situe au même niveau que les autres commandes du langage externe.

**Exemple 1 :** Accès à LISP à partir du langage externe.

>définir fonction ;

Cette dernière commande lance l'interpréteur LISP qui affiche le caractère '*' quand il est prêt. Une fois la commande lancée, l'utilisateur se trouve sous le contrôle de LISP. Il peut dès lors définir ses fonctions.

**Exemple 2 :** Fonction LISP qui retourne le dernier élément d'une liste.

```
*(de LISP dernier (LISP (l))
(cond
((null (cdr l))(car l))
(t(dernier (cdr l)))))
=DERNIER
```

Cette expression permet de définir la fonction 'DERNIER'. Ce qui suit le caractère '=' est le résultat de l'évaluation de l'expression LISP.

### 3.3. Exécution des requêtes

Lorsque le processeur de domaines complexes (PDC) reçoit une requête utilisateur, il va dans un premier temps analyser cette requête afin d'isoler les fonctions utilisateurs nécessitant l'appel à l'interpréteur LISP. Si une fonction est utilisée, il va chercher sa définition dans la base de fonctions et exécuter cette définition sur les données qu'il a sélectionnées. Le résultat de la fonction utilisateur est une liste de tuples contenus dans une relation temporaire. Le processeur PDC envoie ensuite la requête initiale modifiée au processeur suivant.

En reprenant l'exemple 2 qui consiste à rechercher les triangles rectangles dont la surface est supérieure à 100, le processeur PDC recoit la requête suivante :

```
SELECT    *
 FROM     TRIANGLES
WHERE     COORDONNEES.SURFACE > 100
          AND COORDONNEES.RECTANGLE = "Vrai" ;
```

Le processeur commence par chercher les fonctions utilisateurs. Il trouve SURFACE et RECTANGLE qu'il va documenter. Documenter une fonction consiste à demander le code LISP associé. Le code de la fonction est alors chargé dans la mémoire de l'interpréteur LISP. L'attribut concerné par ces fonctions est COORDONNEES. Le processeur demande donc la création d'une première relation temporaire contenant l'attribut clé de la relation TRIANGLES (NT) et l'attribut COORDONNEES. Sur chaque tuple de cette relation temporaire, l'interpréteur LISP évalue les

fonctions SURFACE et RECTANGLE et créé une seconde relation temporaire dont les attributs sont les résultats des fonctions et l' attribut clé de la relation. Le schéma de la seconde relation temporaire est donné Figure 3.3.

| T2 | (NT | SURFACE | RECTANGLE) |
|---|---|---|---|
| | 1 | 50 | Faux |
| | 2 | 30 | Vrai |
| | 3 | 120 | Vrai |
| | 4 | 170 | Faux |
| | 5 | 125 | Vrai |

**Figure 3.3. Schéma et instance de la relation temporaire**

Une fois cette relation temporaire créée, la requête initiale modifiée est envoyée au processeur suivant. L'expression de cette nouvelle requête devient :

```
SELECT   *
 FROM    TRIANGLES, T2
WHERE    T2.SURFACE > 100
         AND  T2.RECTANGLE = "Vrai"
         AND T2.NT = TRIANGLES.NT ;
```

Cette première approche basée sur des techniques de modifications de questions a donc permis de montrer la faisabilité d'une telle intégration. Or les techniques d'analyse et de décomposition des requêtes sont déjà réalisées dans le processeur d'évaluation de questions (PEQ) qui se situe dans la machine assertionnelle. Le but de l'approche future est de descendre une partie de la gestion des domaines complexes dans chaque machine, afin de tirer complètement parti des possibilités actuelles de SABRINA et d'augmenter la puissance d'exécution des fonctions LISP en terme de performance.

## 3.4. Architecture future de SABRINA-LISP

Cette nouvelle architecture se propose d'intégrer la gestion des domaines complexes à tous les niveaux du système SABRINA. La Figure 3.4 présente cette architecture.

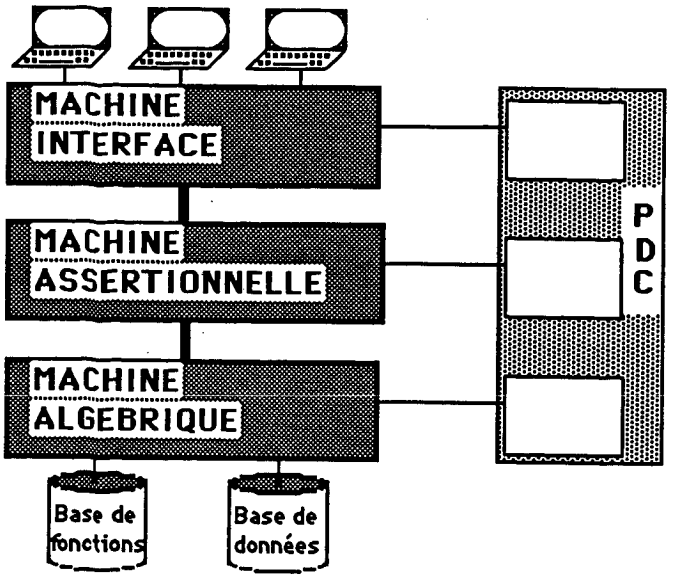


**Figure 3.4. Architecture future de SABRINA-LISP**

Une partie du processeur LISP demeure dans la machine d'interfaces afin de pouvoir accéder au langage LISP à partir du langage assertionnel. L'utilisateur peut définir de nouveaux domaines et de nouvelles fonctions et les stocker dans la base de fonctions. Le processeur LISP accédera au processeur de gestion de la métabase (PGM) pour enregistrer l'existence des nouveaux domaines. Au niveau de la machine assertionnelle, le processeur d'évaluation de questions (PEQ) devra reconnaître les fonctions utilisateurs à l'intérieur d'une requête et décomposera la question en tenant compte de ces fonctions. Le PEQ appellera l'interpréteur LISP pour l'évaluation des fonctions qui retournent une valeur pour chaque tuple. Au niveau de la machine algébrique, un module permet aujourd'hui l'évaluation des fonctions agrégats standard. Ce module sera adapté pour évaluer les expressions agrégats en LISP plus précisément des expressions qui prennent un ensemble de valeurs de tuples en entrée et qui retourne une valeur en sortie.

L'éclatement du processeur PDC dans les trois machines du SGBD est lié aux principales fonctions que doit réaliser ce processeur : interaction avec l'utilisateur, vérification de domaine, calcul de fonctions. Cette distribution des accès au processeur améliore les performances actuelles, et permet surtout de rester cohérent avec l'architecture fonctionnelle de SABRINA et donc de réaliser les mêmes concepts aux mêmes endroits. Cette cohérence va permettre à terme d'améliorer les performances en prenant en compte et en adaptant les structures et les fonctionnalités internes disponibles, en particulier la gestion des chemins d'accès et les algorithmes de sélection, filtrage.

## CONCLUSION

Cet article décrit le SGBD relationnel SABRINA-LISP qui gère des domaines complexes. SABRINA-LISP permet la définition de nouveaux domaines et d'un ensemble d'opérateurs associés à ces domaines sous forme de fonctions LISP. Le principe d'héritage de domaines permet de contrôler l'usage des fonctions.

Un premier prototype de SABRINA-LISP est aujourd'hui opérationnel sur SM90. Il correspond à la première architecture présentée.

Actuellement, deux axes de recherche complémentaires à cette approche sont développés. Les deux axes ont pour but l'amélioration de performances pour ces nouveaux types de données. Il s'agit en premier lieu de pouvoir placer et indexer les relations sur des attributs de domaines complexes et donc de pouvoir utiliser des fonctions LISP dans la définition des arbres de prédicats [GARD 84], [VALD 84]. La seconde approche complémentaire consiste à utiliser un processeur de filtrage intelligent sachant reconnaître des structures hiérarchiques dont la représentation est parenthésée [FAUD 86]. Ces deux études devraient permettre d'améliorer les performances de SABRINA pour le traitement d'objets complexes.

## REFERENCES


[ADIB 85] Adiba M. et al., "Time Concepts for Generalized Data-Bases", ACM Annual Conference, Denver Colorado, Oct 1985.

[ELM 85] Le Maoûlt, R. et Erena L., "Spécifications du langage intégré SINPA", Rapport interne SABRE-CNET, Avril 85.

[FAUD 86] Faudemay P., "Un processeur VLSI pour les opérations de base de données", thèse de l'université Pierre et Marie Curie Paris VI, Juin 86.

[GARD 83] Gardarin G., "Les systèmes et leurs langages", Ed. Eyrolles, 1983.

[GARD 84] Gardarin G., Valduriez P., Viemont Y. : "Predicate trees : a way for optimizing relationnal queries" proc. of the Computer Engineering Conference, IEEE, Los Angeles, Avril 1984.

[GARD 86] Gardarin G., Valduriez P., Viemont Y., Pasquer F., Kerherve B., Simon E., Pastre D., Jean-Noel M., Verlaine L. "SABRINA : Un système de bases de données relationnelles issu de la recherche", TSI Volume 5, no 6, Novembre 1986.

[GOLD 83] Goldberg A, Robson D : "SMALLTALK-80 : the language and its implementation", Edition Addison-wesley 1983.

[GUTT 77] Guttag J.V. et al., "The Design of Data Type Specifications", Current Trends in Programming Methodology, vol IV : Data Structuring, Raymond T. Yeh, ed., 1977.

[NILS 81] Nilsson N.J. "Principals of artifical intelligence", Springer-verlag Berlin, Heidelberg New-York 1982.

[OSBO 86] Osborn S. et Heaven T., "The Design of a Relational Database System with Abstract Data Types for Domains", ACM Transactions of Database Systems, Vol. 11, No. 3, Sept 86, pp. 357-373.

[PASQ 85] Pasquer F.,"Conception et Réalisation d'un Langage de Manipulation de Données Relationnelles à Syntaxe Libre", thèse de troisième cycle, Université de Pierre et Marie Curie - Paris VI, Mai 85.

[STON 83] Stonebraker, M. et al., "Application of Abstract Data Types and Abstract Indicies to CAD Data Bases", Proc. Enginerring Design Applications of ACM-IEEE Database Week, San Jose; Ca., May 1983.

[STON 84] Stonebraker, M. et al., "Quel as a Data Type", ACM,pp. 208-214, 1984.

[STON 86] Stonebraker M., "Inclusion of New Types in Relational Data Base Systems", ACM-IEEE, pp. 262-269, 1986.

[TSUR 84] Tsur S. et Zaniolo C., "An Implementation of GEM: supporting a semantic data model on a relational back-end", Proc. ACM-SIGMOD Conference on Management of DATA, 1984.

[ULLM 80] Ullman J.D., "Principles of Database Systems", Livre, Computer Science Press, 1980.

[VALD 84] Valduriez P., Viemont Y. : "A multikey hashing scheme using predicate trees" ACM SIGMOD Int Conf on Management of Data, Boston, Juin 1984.

[VALD 86] Valduriez P., Khoshafian S., Copeland G., "Implementation Techniques of Complex Objects", Proc. of VLDB, Kyoto, 1986.

[VERS 86] Verso J., "A database machine based on non 1nf relations" rapport de recherche INRIA no 523 Mai 1986.

[WERT 85] Wertz H., "LISP, une introduction à la programmation", Ed. Masson, 1985

[ZANI 83] Zaniolo C., "The Data-base language GEM", Proc.1983 ACM-SIGMOD conference on Management of Data, San Jose, Ca., Mai 1983.

[ZANI 85] Zaniolo C., "The Representation and Deductive Retrieval of Complex Objects", International Conference on VLDB, Stockholm, Août 1985.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique